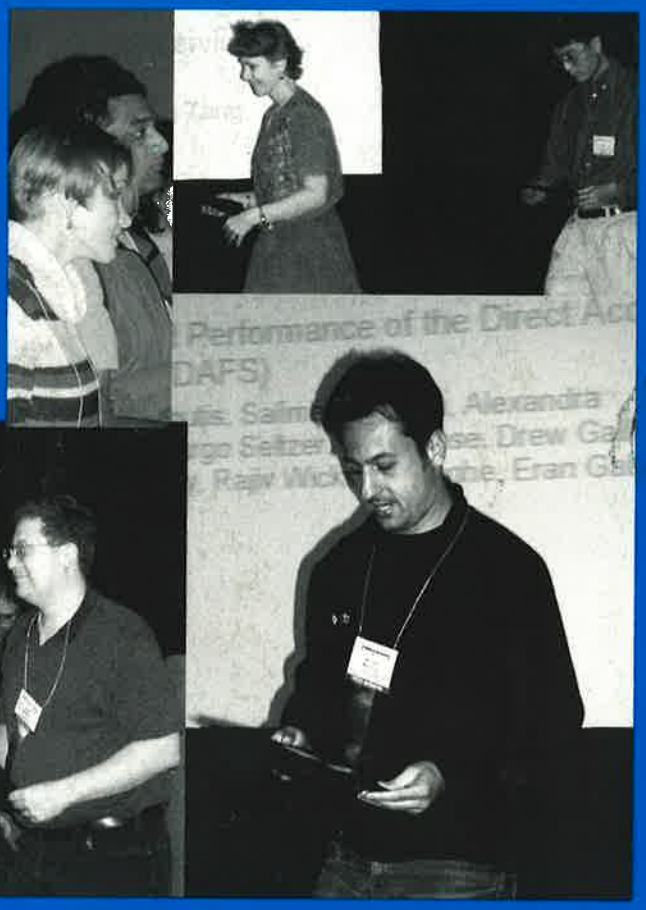
;login:

THE MAGAZINE OF USENIX & SAGE

October 2002 volume 27 • number 5



inside:

USENIX & SAGE

The Advanced Computing Systems Association & The System Administrators Guild

USENIX Upcoming Events

16th Systems Administration Conference (LISA '02)

Sponsored by USENIX & SAGE

NOVEMBER 3–8, 2002
Philadelphia, Pennsylvania, USA
http://www.usenix.org/events/lisa02

Internet Measurement Workshop 2002

Sponsored by ACM SIGCOMM, co-sponsored by ACM SIGMETRICS and USENIX

NOVEMBER 6–8, 2002
Marseille, France
http://www.icir.org/vern/imw-2002/

5th Smart Card Research and Advanced Application Conference (CARDIS '02)

Sponsored by USENIX and the IFIP Working Group 8.8 (Smart Cards)

NOVEMBER 20–22, 2002
San Jose, Califonia, USA
http://www.usenix.org/events/cardis02

2nd Workshop on Industrial Experiences with Systems Software (WIESS '02)

Sponsored by USENIX
Co-sponsored by ACM SIGOPS & IEEE TCOS

DECEMBER 8, 2002
Boston, Massachusetts, USA
http://www.usenix.org/events/wiess02

5th Symposium on Operating Systems Design and Implementation (OSDI '02)

Sponsored by USENIX
Co-sponsored by ACM SIGOPS & IEEE TCOS

DECEMBER 9–11, 2002
Boston, Massachusetts, USA
http://www.usenix.org/events/osdi02

5th NordU/USENIX Conference (NordU2003)

Sponsored by EurOpen.SE and USENIX

FEBRUARY 10–14, 2003, Västerås, Sweden
Web site: http://www.nordu.org/NordU2003/

4th USENIX Symposium on Internet Technologies and Systems (USITS '03)

MARCH 26–28, 2003
SEATTLE, WASHINGTON, USA
http://www.usenix.org/events/usits03
Notification of acceptance: November 8, 2002
Camera-ready final papers due: January 17, 2003

2nd Conference on File and Storage Technologies (FAST '03)

MARCH 1–APRIL 2, 2003
San Francisco, California, USA
http://www.usenix.org/events/fast03
Notification of acceptance: November 1, 2002
Camera-ready final papers due: January 6, 2003

The First International Conference on Mobile Systems, Applications, and Services (MobiSys '03)

Jointly sponsored by ACM SIGMOBILE and USENIX in cooperation with ACM SIGOPS

MAY 5–8, 2003
San Francisco, CA, USA
http://www.usenix.org/events/mobisys03/
Notification of acceptance: December 18, 2002
Camera-ready final papers due: March 4, 2003

2003 USENIX Annual Technical Conference

JUNE 9–14, 2003
San Antonio, TX, USA
Paper submissions due: November 18, 2002

12th Usenix Security Symposium

AUGUST 4–8, 2003
Washington, DC
Paper submissions due: January 27, 2003

For a complete list of future USENIX events, see **http://www.usenix.org/events**

contents

;login: Vol. 27 #5, October 2002

;login: is the official magazine of the USENIX Association and SAGE.

;login: (ISSN 1044-6397) is published bimonthly, by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$50 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $60 per year.

Periodicals postage paid at Berkeley, CA, and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Cover: Best Paper award winners at USENIX: clockwise from top left: Alexandra (Sasha) Fedorova and Rajiv Wickremesinghe; Ludmila Cherkasova and Wenting Tang; Kostas Magoutis; Sotiria Lampoudi and David M. Beazley. See pages 62ff.

motd

by Rob Kolstad

Rob Kolstad is currently Executive Director of SAGE, the System Administrators Guild. Rob has edited *;login:* for over ten years.

kolstad@sage.org

Leverage

I keep thinking that I will soon be wizened and know everything I need to know. I avoid prejudices and try to judge actions and results more than hype.

I try to partake of technology services so that I can respond intelligently about Web sites, software services, initiatives, ideas, and the like. I pay my bills online. I read my email. I installed a spam filter (what a treat; I am using SpamAssassin and it's so much better than not having it). I have online access throughout my house via strategically placed laptops and other computers. I have both UNIX(-like) and Windows running on various platforms both fixed and portable.

I had the opportunity last week to extend my knowledge of the use of Windows. I had a document that was required to be formatted "nicely" in order that I might distribute it to SAGE members who might wish to see it. For better or worse, the document contained roughly 40 tables and a dozen postscript graphs. Troff is not regarded as the world's best table formatter, so I embarked on implementing something a bit snazzier that would inspire my readers to enjoy the document and its myriad figures.

I had implemented the original document in HTML but found I needed a PDF document. We know that HTML → PDF converters are not renowned for their beauty. I tried to get the document typeset (by someone else) in Quark but initially failed on a quick success due to scheduling and other non-technical reasons. So, I reasoned, Quark can read MSWord files, I'll just format the document in MSWord and then it will be trivially convertable to Quark.

I spent a few minutes writing a short script to extract the document from the HTML so that it could be imported into MSWord and formatted (for the text) and converted to a table (for the tables). After accidentally typing the wrong filename, I learned that MSWord will read HTML directly and do a credible job of laying it out – including tables! I figured my troubles were over.

That was before my ultimate epiphany about MSWord (and, as it turns out, so many of the "tools" under Windows). I found that while MSWord has styles and templates for global formatting conventions, it is difficult to get tables to fall into a consistent format. Each table required personal attention. Even moving the tables to be justified on the right margin was also not possible without clicking on each table and moving it. In fact, I ultimately spent about 25 hours trying to get the document to be passably presentable.

What was the epiphany? Windows has the wrong kind of leverage. I think of leverage as using devices (e.g., computers) or people to increase one's reach or one's ability to solve problems. The Windows tools I was using were insidious. They accepted incremental improvement but never enabled me to improve by an increment of any appreciable size. I could inch closer to my desired solution, but only a little at a time – a snail's pace. "I think I'd like the tables on the right margin," required multiple clicks on each table. Argh!

Why is it insidious? Because it was always possible to move (often to move closer to the solution). At no time did I fail to have some (potentially untried) option or some idea of how to move forward. I have every reason to believe that people who format one table every two months find extraordinary power in the ability to do so with MSWord.

And, to be fair, MSWord works for dozens (hundreds?) of languages. The folks at the international programming contests regularly translate the problems into 65 languages – all using MSWord. This is an incredible feat!

But I hate it. No leverage! More skill or practice only incrementally improves my ability to format tables (in this case). The same seems to be true for any repetitive operation that one would like Windows to perform (e.g., "Please convert these 22 PostScript files to PDF" for our LISA attendees). That's 22 sets of choose-a-file, convert, confirm, and iterate. Ick.

So, I am now familiar with the state of the art. It was an epiphany for me.

apropos

I'll Scratch Your Back . . .

by Tina Darmohray

Tina Darmohray, co-editor of *;login:*, is a computer security and networking consultant. She was a founding member of SAGE. She is currently a Director of USENIX.

<tmd@usenix.org>

A few years ago I wrote an article entitled "Ready, Set, No?" in which I outlined things that I thought a good manager does for – and doesn't do to – their employees. Apparently it struck a chord with many readers, since I continue to receive comments about it.

Initially, I was surprised at the response, but it just underscores that most of us want a few very similar things from our managers. We want our managers to set the tone for the work environment. They should provide leadership, but also mentoring and learning opportunities to let the employees know that they are valued today, and will be in the future as well. A savvy manager will figure out what matters most to each employee and, wherever possible, make things like work assignments, office arrangements, and face-time hours accommodate those preferences. Managers shouldn't set their employees up to fail with unrealistic workloads, deadlines, or expectations. And when politics, heavy workloads, or other office snafus inevitably rear their ugly head, managers who support their employees, run interference for them, or simply push back a little, will gain employees' undying loyalty and their willingness to go the extra mile – after all, "s/he did it for me."

What about the individual contributor? Are we all created equal? Or can our performance in the workplace vary, and if so, what makes a good worker? I asked a few friends of mine about this and found that, just like managers, there are soft skills that make individuals better to work with and land them consistently on the top of the heap among peers and managers alike.

Whenever the opportunity presents itself to make your boss or co-workers look good, do so! This has multiple payoffs. You'll be rewarded for your individual contribution, but you'll also be recognized for the part you play in giving credit to others. There is usually ample praise to go around, so make yourself look good by making others look good too.

If you want your technical prowess to really shine, share it with others! Those employees who inform others about newsworthy items, teach about and document what they're working on, and mentor peers are much more highly valued than those who don't. If you have a clue, don't keep it to yourself! Teaching and mentoring are forms of leadership and are often rewarded come advancement time.

If you see the wreck coming, telegraph the information ahead of time. No one likes to be the bearer of bad news, but if you feel your manager is going to be blindsided, it's better to give an in-private "heads-up" than let them be publicly caught off guard. No matter how bad the news, they'll appreciate hearing it and appreciate you for giving it.

There's more to most business decisions than simply technical input. Managers are tasked with considering a larger picture, such as risk assessment, profitability, and return on investment. Employees should provide their managers with all the technical information necessary for these big-picture decisions and be willing to entertain give-and-take discussions that involve non-technical considerations. Employees who demonstrate the ability to see the whole picture will be called upon more often for advice, and will ultimately be in a better position to have it adopted.

We expect a lot of our managers, but it's not a one-way street. Make your manager look good, share a clue, and get behind the big picture. The sooner you get on board with these soft skills, the sooner you'll become that all-around employee who has their manager's ear.

letters to the editors

To John Nicholson:

I thoroughly enjoyed your article, "Politeness in Computing," which appeared in the February 2000 edition of *;login:*. In particular I appreciate the commonsense analogies you have drawn to a home with a welcome mat. You have a gift for explaining difficult technical/legal issues. Have you written any similar articles discussing the legalities of "honeypots"? I am the InfoSec security specialist at my firm. Along with my superiors and the legal department, we are having a lively debate after reading an article by William Jackson in Government Computer News, *http://www.gcn.com/vol1_no1/daily-updates/19506-1.html*.

He basically cites a speech by one Richard P. Salgado, a trial attorney in the [Justice] Department's Computer Crime and Intellectual Property section, in which Mr. Salgado preaches caution. One of our attorneys feels that the entrapment defense is a "pure fantasy" and I am inclined to agree, but others are more cautious. The downside of being cautious is that it tends to make my job of protecting information assets more difficult.

I eagerly await any input you may have.

Regards,

Dave Warde

John Nicholson replies:

Dave –

I'm glad you liked the article, and, just as importantly, my editors are glad you liked the article. It's funny that you should ask about honeypots, because that was the issue that got me into the writing business in the first place. There was an argument a couple of years ago on one of the Security Focus lists about whether honeypots are entrapment or not, and one of the *;login:* editors saw my response and asked if I would be willing to write a column.

To address the issues raised in the article:

Federal wiretap laws prohibit interception of electronic communications, including traffic monitoring across a network. There are exceptions for network protection, but Salgado said that is an "uneasy fit" for honeypots, because they are set up with the expectation of being attacked.

This isn't entirely correct. If you are the owner of the network, you can monitor what happens on it. You can doubly protect yourself by putting a banner on your login page that says that any use of the network is subject to monitoring, but the key thing that courts have looked at with regard to such monitoring is whether the person had a legitimate expectation of privacy in the communication. I think a judge would have a tough time accepting an argument that someone attacking your network had a legitimate expectation of privacy in his/her attack.

Even if you were only allowed to monitor your network for defensive purposes, I think the honeypot could arguably qualify as a defensive tool. For example, I have a limited budget for physical security at my home. I recognize that there are a number of ways that someone could break in, and I take steps to secure or prevent those. However, if someone is determined to break in, I must recognize that they will find a way. To deal with that possibility, I try to recognize where an intruder might be able to break in, and I have cameras in those areas. If I could only afford a certain number of cameras, I might make one path a little easier or more attractive than the others so that the intruder would take that path and thereby pass in front of the camera, allowing me to gather evidence of the crime. The intruder has already commit-

letters to the editors

ted the crime by being inside the house; the camera simply collects the evidence. By placing a honeypot and monitoring it, you are simply putting an intrusion detector on a place where unauthorized individuals are likely to go, if they are already committing the crime of being inside your network without authorization.

An operator might be held liable for damages if a compromised honeypot is used to launch an attack against a third party. "We don't know" if such liability would hold up in court, Salgado said.

This is theoretically possible, and I actually wrote another article for *;login:* on this subject called, "You've Been Cracked . . . And Now You're Sued." But if you're setting up a honeypot, you ought to be sophisticated enough to isolate it and prevent outbound attacks on other networks (or at least either notify those networks that they are being attacked or shut down the attack as soon as it starts). There's really no excuse for setting up a honeypot and then allowing it to be used as a zombie.

A hacker charged with illegal activities involving a honeypot could argue entrapment, which Salgado said is a difficult defense. He said it might not apply to so-called passive honeypots.

Salgado is correct that entrapment is a very difficult defense. The article doesn't point out, however, that the defense of entrapment is also only available to someone who is being prosecuted as the result of activity by a government agent (like the DOJ, FBI, or some state or local law enforcement agency). If your company (or client), as a non-governmental entity, sets up a honeypot and a cracker gets prosecuted because of it, the defense of entrapment is not available. See the legal definition of entrapment at *http://dictionary.lp.findlaw.com/scripts/results.pl?co=dictionary.lp.findlaw.com&topic=64/64a96fc79d0fff3a77e4ddea401c7688.*

Furthermore, as Salgado also notes, because a honeypot is a purely passive thing, even if you were a government agent, you are not really inducing or encouraging a potential cracker to go attack it. If you are a government agent and set up a honeypot and then anonymously went to hacker sites and talked about this fantastic server with all kinds of really cool stuff on it and how easy it was to own, etc., etc., then you might be setting yourself up for the defense of entrapment.

Hope this helps. Feel free to write back with any comments or questions.

John.Nicholson@shawpittman.com

Correction

Robert Faust's article in the last issue (Vol. 27, No. 4, August 2002) contains an error: the last equation on page 41 should read:

$$\textit{Received signal strength (non-normalized \%)} = .83\left(C - 35\log_{10}\sqrt{(A-x)^2 + (B-y)^2}\right) + 83.891$$

vive la révolution! now get over it!

by Strata R. Chalup

President, VirtualNet. Starting as a Unisys 68K admin in 1983, Strata Chalup is now an IT project manager but allegedly has retained human qualities. Her mixed home network (Linux, Solaris, Windows) provides endless opportunties to stay current with hands-on tech.

strata@virtual.net

Let's consider the small-appliance manufacturing industry. I sincerely doubt that anyone goes into that industry expecting to comfortably retire a few decades early by dint of hard work and picking the right company. I also doubt anyone there expects to be lionized in glossy self-referential magazines or considered an "industry elder statesman" for the supreme achievement of starting a can-opener or toaster company and managing not to go out of business for several years.

We need to abandon the fiction of the high-tech industry as "the future." It is not the future. It is merely part of the future, yet another tool in the growing plethora of tools developed by civilization. The revolution is over. Technology won. As technologists, we won. We just may not have realized it yet.

After Enlightenment, Carry Water, Chop Wood

Now you turn to me with incredulity and say, "How can you say that we 'won' the computer revolution?! For Turing's sake, more people than ever are using Windows, the Internet is full of spam, Congress is passing all these ridiculous laws about stuff they clearly don't grok, and SAGE is still chafing under the iron yoke of USENIX! Okay, maybe that last one isn't the big deal some people make it out to be, but what about the other stuff?!"

To this question I must pose another: what happens when you win a revolution? There are several large-scale effects, of which two are most noteworthy. Firstly, the patriotic militia now must lay down its arms and go back to being shopkeepers, farmers, and draymen. Think of the American Revolutionary War (or Rebellion, with a cordial nod to our neighbors across the Pond). The minutemen who survived went back to what was left of their shops or farms and got back to work. Not very glamorous.

Secondly, and perhaps more applicably, when society-at-large accepts the validity of a general concept, that acceptance is far from complete and universal; it is often without full comprehension and always comes with some highly vocal dissenters. In sports terms, players take the ball and run with it – but probably out of bounds and often along a playing field that looks nothing like what the game's originators imagined. General societal adoption of technology takes "our ball" and goes off to play with it according to larger rules: in particular, with adaptation of technology into more specialized tool sets and re-purposing of technology and technological infrastructure for previously unfeasible uses – including those which we, as de facto technocrats, may feel lack a certain elegance or relevance.

Which Came First: The Goose or the Golden Egg?

To make matters more confusing, our technological revolution has largely become conflated with a minor economic "revolution" – in the old-fashioned sense of "a turn of the wheel." The economic boom cycle came to be considered part and parcel of the technological progression. The success of the technology boom cycle was the widespread normalization of an unprecedented level of technological sophistication. Today's high school kids tweaking their Doom skins have more power to do graphics visualization than the entire MIT astrophysics department had in 1981. Wow. The guy on the street doesn't think twice about this anymore. Double wow. To us it seems to matter that the kid may be running Windows instead of Linux, but in the larger context, the victory is that he or she is using a sophisticated tool and not thinking about

OS or hardware issues at all. Let's not forget the forest as we argue about which trees are better, and which types of woodlot management are best.

Did you know that not only your box-o-hardware but your OS and your applications are now commodity items? Why do you think Microsoft wants to turn on system auditing with mandatory updates in Windows 2000 (SP 3, read the EULA) or application licensing in XP? Commodity, commodity, commodity. Applications are just conduits for file formats, which in turn are just a convenient box with handles around *content*. "Pththt!" you say. "Content is king? Been there, done that, no market in it." Not quite true – there's no content market with pie-in-the-sky valuations that will catapult a company to stardom. What there is, instead, is a vast and insatiable long-term income stream. The big companies are spending millions twisting our legislative structure to support this income stream, this multi-billion-dollar market for those who aggregate intellectual property rights and copyrights to content.

Day-to-day technology is rapidly becoming merely a conduit to deliver content to those who require it. This mirrors the business world, where technological applications have been largely harnessed to make existing business processes easier. Everyone who has used it has a Meeting Maker™ horror story, yet for the corporation overall it still makes a positive difference. The success of the PDA is largely rooted in synchronization, and could not come to fruition until there was enough electronic infrastructure carrying out business processes that there was something on the other end to synchronize *to*.

I Will Gladly Pay You Tuesday for a Hamburger Today

This brings us to another key point. What many of us in the technology field tend to forget is that technology has everyday value only when it is *applied*. Further, Moore's Law gives technology something of the nature of a perishable good or even a service. This may seem counterintuitive, given the tangible nature of a rackmounted box stuffed with hard drives and application servers. The economics of perishable goods and of services are rather well understood. Empty hotel rooms are not bankable. The high prices with which we are familiar, and upon which empires were built, came largely from relative scarcity of technology applied in the right place at the right time. When you just *had* to have a 100GB RAID 5 file system to support your e-commerce application, and the server box only had so many disk drive slots, a 10GB 9600 rpm drive was priceless. Especially when time to market was perceived as *the* most critical factor, and a dozen other dot-com companies were trying to pry the same pallet of in-stock drives out of your local vendor's warehouse. What if there weren't a dozen other companies trying to buy the same drives? Or fast-forward six months to a year, when the same dollars buy you a 30GB drive. Those 10GB drives sitting on the shelf still cost the same to produce but their dollar value has dropped. Yet we consistently see business plans that insist that the price should have nothing to do with the current situation, and that the "value" of the drives is $whatever.

This is an understandable, yet common, mistake. I had hopes that an MBA and a little bit of history would have proven more of a deterrent. Most of us think of technology prices as intrinsic rather than situational. The cost of manufacturing technically does not include research and development, despite our vendor's sales team's assurances to the contrary. Hence the thriving clone markets and the profitability therein. They're not building empires, but they're paying salaries and staying in business. It's a brave new world to the people who entered the workforce in the late '80s, but it's "back to reality" for the rest of us. In my most cynical moments I think that the industry will

Technology has everyday value only when it is *applied*.

Plenty of successful companies are making can openers. They don't have stock with valuations reflecting obscene P/E ratios.

resist normalization until everyone who remembers the glory days of IBM and DEC retires from positions of responsibility. We may have to broaden that to include Sun, Microsoft, and Oracle. The time of empires has passed. When the market was smaller, you could corner it. Who has cornered the market on butter? On best-selling novels? On gasoline?

It's the End of the World As We Know It . . .

Much of the technology sector is waiting for investors to "come to their senses." Sorry, they already have! Except for random islands of scarcity produced by the market equivalent of "lift pressure," the high flying days are over. Plenty of successful companies are making can openers. They don't have stock with valuations reflecting obscene P/E ratios. After all, they're making can openers. Welcome to the commodity marketplace.

In many ways the burst of the dot-com bubble was like Watergate. Its worst aspect was not the individual investments or reputations destroyed, but the damage to the societal perceptions of the institutions behind them. There are still many things that we can do with the new technologies, but the man-on-the-street perception in much of our society today is that *all* technology is over-hyped and that there's nothing out there but Web site pyramid schemes and pop-up ads.

There are still good technology stocks out there. They're the kind that will help fund your retirement in 40 years, not the kind that will buy you a house in five years. The familiar proverb is missing a key modifier – it's not that "there's no such thing as a free lunch"; it's "there's no such thing as a *consistently* free lunch." If you're used to having Peets or Starbucks in the break room, "Office Caterer House Blend" seems like a deep pit rather than a return from the mountains to sea level.

. . . And I Feel Fine

Do you remember life before Post-it™ Notes? A lot of us can't imagine being without them. Their invention didn't "make" 3M, nor did they become 3M's flagship product against which all other products were measured. Their inventor didn't leave 3M and try to start his own company to propel himself into stardom and riches. There is still a lot of potential out there. It's just *traditional* potential, which can feel a lot like "nothing" in contrast with the excitement of the past couple of decades.

So where is this all going? Is the high-tech sector dead? Should we all just take up dog walking or open a hot dog stand? I think that the end of the Age of Empires in high tech has a lot of potential for us as a profession. Work can still be fun, which is good because you won't be retiring at 35. Opportunities are out there, and while they aren't as glamorous, they may not demand as high a personal price in unpaid overtime and ulcers.

There are still next-level paradigm jumps out there. They may not make you rich, but who cares? As the late Michael Dertouzos said at a futurism forum in 1999, "We haven't really invented the bulldozer yet. We're just out there with platinum, diamond-encrusted shovels." I think that there's the potential for a New Internet that looks a lot like the Old Internet, a place where innovation was exciting and people did it for fun, and innovators weren't punished with a four-digit surprise bandwidth bill from their ISP. New advances might make hardware hobbyism possible on a level that would remind old-timers of the heady days of the '70s when "home computing" meant breathing solder fumes. Details, details, you say. But that's the next column. See you next issue!

software licensing 101

by John Nicholson

John Nicholson is an attorney in the Technology Group of the firm of Shaw Pittman in Washington, D.C. He focuses on technology outsourcing, application development and system implementation. and other technology issues.

John.Nicholson@ShawPittman.com

We've all seen them – pages and pages of two-column text, generally in 8-point type. In real life, we might get a CD in a jewel case with lots of small text printed on it and a warning that says that opening the box means that you accept the license. When we're online, we get an easy "I Accept" box that doesn't require reading all the gibberish before letting us install the software that we want. But have you ever actually read one? What are you agreeing to? Software vendors have lawyers who spend hundreds, if not thousands, of hours drafting these licenses to protect the "rights" of the seller. And in the case of Microsoft, Adobe, AutoDesk, Borland, and several others, they also have the Business Software Alliance (*http://www.bsa.org*) to collect reports of and prosecute unauthorized software use.[1] The only thing protecting your company is your understanding of the terms of the license and your (or your lawyer's) ability to negotiate with the vendor.

The purpose of this article is to give you some insight into what a software license should cover, provide an analysis of some sample language that vendors have used, and give you some tips for negotiating with vendors.

What Is a "License"?

The word "license" is somewhat misleading. A software license is a contract. According to WhatIs.com, "An End User License Agreement (EULA) is a legal contract between a software application author or publisher and the user of that application. The EULA, often referred to as the 'software license,' is similar to a rental agreement; the user agrees to pay for the privilege of using the software, and promises the software author or publisher to comply with all restrictions stated in the EULA."[2] So, in exchange for being allowed to use the software, you agree to comply with the restrictions imposed by the vendor. If you break the rules, you have breached the contract, and there may be specific remedies written into the license or you may have even violated some laws (e.g., copyright laws,[3] the Digital Millennium Copyright Act,[4] the No Electronic Theft [or NET] Act,[5] or others).

License Structure and Analysis

A software license tells you what you can and can't do with the software and what the vendor can and can't do. What the license says (and doesn't say) is just as important to you as it is to the vendor. In general, a software license should cover:

- Software included under the license
- Scope of use for the software and any restrictions on its use, as well as any different restrictions on the use of the documentation
- Duration of the license
- Related services (e.g., consulting, enhancement, help-desk support) that will be provided, and the terms under which those services will be provided
- Pricing and payment terms
- Confidentiality provisions
- Warranties and indemnities
- Limitations on liability
- Termination of the license and/or the services
- Other legal terms that are relevant (e.g., rights of publicity, choice of law)

Frequently, software licenses drafted by vendors will try to limit software use to employees of the company purchasing the software.

If the software is going to be used internationally, there may be specific terms that need to be discussed. For example, the US government restricts certain software from being exported to certain countries or at all, and you might have to apply for a special permit; or there are certain international treaties that may come into play regarding the sale of goods.

Software Included Under the License

The first question to ask when looking at a software license is whether it covers all of the software (and all of the functionality) that you think you are getting. It's important to make sure that you understand how the software covered by the license will provide *all* of the functionality that you think you are getting, and that it will continue to provide the same functionality in the future. Although this may seem obvious, there are certain provisions related to updates and upgrades that can affect the availability of functionality down the road. (See the section on updates and upgrades, below.) When you are discussing the software covered by the license, be sure to review in detail how the software will provide the functionality that you think you are getting. If the software cannot currently provide some of the functionality that you need, see whether there is a workaround, whether the vendor plans to provide such functionality in the future, or whether you will need to develop that functionality in cooperation with the vendor, using a consultant or on your own.

Scope

The scope of a license can vary dramatically, depending on the restrictions that the vendor wants to place on your use of the software. The license may specify:

WHO CAN USE THE SOFTWARE

Frequently, software licenses drafted by vendors will try to limit software use to employees of the company purchasing the software. This would mean that neither your contractors nor any consultants could use the software, which could be an inconvenience for you (if you complied with the restriction) or could result in a breach of the license (if you didn't). If you are only using object code, there isn't really much reason for this restriction. If you have source code, it would be reasonable for the vendor to insist that contractors and consultants sign a Non-Disclosure Agreement before being allowed access to the source code. And there might even be a short list of direct competitors that the vendor would want to prevent from seeing the source code under any circumstances.

FOR WHOSE BENEFIT

Software vendors want to sell as many copies of their software as possible, so they try to limit the number of people who can benefit from each installed version of the software. You, on the other hand, want to buy as few copies of the software as possible, so you want to be able to use it for your own company, your subsidiaries, other affiliates, your customers, your suppliers, etc. As long as you and the vendor come to an understanding regarding how the software will be used prior to executing the license, the vendor can take the scope of your use into account in its pricing.

FOR WHAT PURPOSES

Just as vendors want to limit access to the software by other companies, vendors want to limit access across business lines, if possible. By understanding how your company will want to use the software, you can reach a reasonable agreement with the vendor in this area.

IN WHAT LOCATIONS

If you plan to use the software in multiple countries, some vendors will try to argue that they need to get approval from regional segments of their organization in order to negotiate a deal. How the vendor organizes its own accounting is not your problem, and if a vendor wants to be a global player, it should act like one. (Note: this is just an extension of the "no authority" strategy discussed in the "Negotiation Tips" section later in the article.) It is possible, however, that there may be countries in which, for some legitimate reason (generally legal or regulatory), a vendor is unable to license you to use the software.

ON WHAT EQUIPMENT

Certain licenses are restricted to a particular-sized processor. Be certain that the license you are getting is sufficient for the use you intend now and throughout the term of the license. You will not put yourself (or your successor) in a good negotiating position if your business needs to renegotiate the license in the middle of the term because you did not adequately anticipate growth levels. If the pricing for the license is based on the processor (or other equipment) that you will be using, and there is a possibility that you might need to upgrade or expand your equipment during the term, negotiate the pricing for using the software on the upgraded/expanded equipment before signing the initial deal. Remember, your negotiating leverage is at a maximum before you sign.

Certain licenses are restricted to a particular-sized processor.

If the license is based on number of users or number of machines accessing the software, you should consider whether devices like PDAs would qualify. For example, one vendor's standard license prices the software by "client," which it defines as "an application that invokes, typically via a network protocol, the software functions provided by one or more servers." Under that language, a query from an application installed on a PDA could qualify as a "client," which could dramatically increase the price of the software.

PERMITTED COPYING

Vendors frequently include language such as, "Customer shall not make any copies of the Software, except for a single archival copy solely for internal purposes." This means that, for example, you would have to separately license development, test, and production instances of the software, and that you might also need to separately license any copies to be used for training. Since you must have at least a test region in addition to your production region, the license should include copies for you to use in those additional regions without an additional charge. Furthermore, if you have multiple locations, each running the software locally, you might want to have a backup copy at each location so that you can re-install via the local network. This is also a reasonable request.

Another thing to look out for in software licenses is whether the defined term "software" includes the documentation. Frequently, it does, and the impact of that definition combined with the language quoted in the previous paragraph would mean that you could not copy any of the documentation associated with the software in order to make training materials or procedures manuals. Training materials and other documentation provided by the vendor are likely to be generic rather than customized to your particular implementation. To customize your documentation, you might want to make sure that the license includes language such as, "Customer may copy, share, distribute, modify and create derivative works from the user manuals and any related documentation solely for Customer's internal business purposes."

WHETHER THE LICENSE IS TRANSFERABLE OR NOT

Frequently, a license will specify either that (1) the customer may not assign the license without the vendor's consent or (2) neither party may assign the license without the other party's consent. If your company is acquired or wishes to assign the license to another company (including a parent, affiliate, or subsidiary) for some other valid business reason, this provision could allow the vendor to refuse to grant its consent unless you satisfy conditions specified by the vendor. Language like this should be revised so that you can transfer the license to a parent, affiliate, or subsidiary without being required to get the consent of the vendor. At the least, you should get the vendor to agree that it will not unreasonably withhold any consent.

Duration/Termination of the License

The license should specify the term of the license. Large system operating software is generally priced by the month or the year. In the US, a perpetual license is legally acceptable, but outside the US you should discuss with a lawyer using a term of 20 years of less in order to avoid certain legal issues related to perpetual contracts.

In addition to knowing when the license will expire, you should examine very carefully the ways that the license can be terminated. A piece of software can be critical to your company's operations. Unless you don't pay for the license or you knowingly disclose the vendor's legitimately confidential information, the vendor should not be able to terminate the license. There are legal (i.e., financial) remedies that can compensate the vendor for virtually anything else that your company could do without threatening your company's ability to use a potentially mission-critical piece of software. On the other hand, you should be able to terminate the license and associated maintenance as long as you provide a reasonable amount of notice to the vendor.

Frequently, vendors will agree to allow you to terminate a contract early provided that you give them notice and pay a termination-for-convenience fee.[6] These fees range from a reasonable recovery of costs by the vendor (such as any investments that the vendor had to make to provide your particular services or to develop a particular function) to bordering-on-absurd attempts to include all of the revenues that the vendor would have received during the term of the contract. If you terminate the contract early, it isn't unreasonable for a vendor to recover costs that the vendor expected to recover during the term. The vendor shouldn't suffer harm because you elected to terminate a contract early. However, once you terminate the contract, the vendor won't be providing services or incurring costs, so there is no reason for the vendor to receive the total revenues that the vendor would have received for providing the services during the remainder of the term.

Nor is there much ground for the vendor to argue that it should receive the profits that it would have received during the remainder of the term. Vendors will try to use the argument that they have a right to lost profits because they shouldn't suffer harm because you decided to terminate the contract early. If the contract had never happened, the vendor would not have received the profits that they are trying to claim, so allowing the vendor to include lost profits in a termination-for-convenience charge puts the vendor in a better position than if the deal had never happened. The vendor should be allowed to recover costs, but not lost profits.

Payment, Acceptance Testing, and Warranties

In general, the vendor wants its money for the software right away (particularly these days). When negotiating, vendors look for payment on delivery (if not sooner) and in

a lump sum. For example, one vendor's standard license agreement specifies, "Within 10 days after the Effective Date of this Agreement, the applicable Order Form and payment to [Vendor] of the associated license fees, Support Fees or other fees set forth in the Order Form, [Vendor] shall provide access to the [Vendor] FTP site to enable licensee to download the license Products and Documentation." This means that you have to pay for everything up front and only then will you be able to download the software and documentation. For something really simple and basic, this might not be too bad.

But for more complicated software that will require some implementation, you should be able to delay at least some of the payment until the software is successfully implemented (i.e., after acceptance testing). Vendors will argue that acceptance testing isn't necessary because (1) the product is proven in the market and (2) you're getting a warranty. However, the warranties frequently are effective for 90 days after *delivery* of the software. Most implementations take longer than 90 days, so unless the implementation is either simple or on a fast track, your warranty is effectively useless if it is based on the delivery date. Your warranty should begin running after the software has been successfully installed and tested (including integration testing), and you should be able to return the software for a full refund of the license fees if it isn't accepted.

Vendors want to be able to recognize the revenue from the sale of the software as soon as possible, and allowing you to return the software can interfere with the vendor's ability to recognize the revenue. Revenue recognition affects the vendor's earnings and the salesperson's incentive compensation. There are very specific rules regarding when revenue can be recognized.[7] If revenue recognition becomes an issue in your deal, you should have someone from your accounting/finance group who is experienced in this area working with your team.

For ongoing services or installment payments, you should look at the payment terms specified in the license. For example, one vendor's standard terms states:

> All billed charges are due ten (10) days from the invoice date. Licensee agrees that a copy of an invoice received by facsimile machine shall be binding on Licensee and have the same effect as an original. All balances ten (10) days past due will be subject to a ten (10) percent annual finance charge, and [Vendor] may elect to suspend technical support, software updates and enhancements, withhold shipment of computer supplies, and/or activate time sensitive devices . . . until [Vendor] receives said past due payments.

Not only is "Net 10" a short period of time for your company to make its payments, but the multiple remedies to this vendor of being able to charge interest, withhold support or other services, *and* potentially disable the software for failure to pay a bill within 20 days of the date the invoice was printed (not received by your company) is unreasonable. Because the license specifies an interest charge for late payments, the vendor is already being compensated for a late payment.

Remote Deactivation of Software

As mentioned in the language quoted in the previous section, vendors sometimes include in licenses the right to switch off the software or deny you access to date-sensitive activation codes if you haven't paid for something or if the vendor thinks that you are in breach of the license. For example, one vendor's standard form license states:

You should rarely, if ever, accept a license that allows the vendor to deactivate any part of your software with time-sensitive disabling devices.

> Licensee acknowledges that some or all of the [software] may contain time sensitive devices which may be activated automatically, by [Vendor] or otherwise upon a material breach of this Agreement by Licensee, including without limit Licensee's breach of the payment terms . . . and Licensee's breach of the confidentiality provisions . . . or the expiration or termination for any reason of this Agreement. Upon activation, such time sensitive devices may alter or prevent the functionality of the [software]. Licensee acknowledges and agrees that such time sensitive devices are necessary to protect [Vendor's] intellectual property rights, that [Vendor] shall have no liability whatsoever for any outcomes caused by activation of such time sensitive devices and that Licensee shall be liable for all costs associated with the activation of such time sensitive devices, as well as costs associated with resuming normal use of the [software].

You should rarely, if ever, accept a license that allows the vendor to deactivate any part of your software with time-sensitive disabling devices. If the vendor can deactivate your software, you will likely give in to the vendor's demands rather than lose your functionality. Under the license, and under the law in general (at least in the US), the vendor will have legal remedies for any breach; the ability to remotely disable the software represents a self-help measure that gives the vendor unreasonable leverage over your company.

Maintenance and Support

General

The standard vendor position regarding maintenance and support is that it begins after the warranty period (if any), is payable in advance, is non-refundable, and is subject to annual increases that are not capped in any way. From your perspective, you want the maintenance and support to be optional, since you might not need it or might be able to get it from someone else; you want to be able to pay in installments; and you want the price to be fixed during the term of the license, or if not fixed, then at least have the increases specified in advance and capped, to ensure that you understand the maximum amount of costs that you will have to pay.

New Releases, Updates, and Upgrades

Frequently, vendors will include new "releases," "updates," or "maintenance patches" in the cost of maintenance but will expect you to pay for "upgrades" to new "versions." Each vendor uses these terms differently and, frequently, in confusing ways. You should understand how the vendor numbers its software, and agree in advance to what constitutes something covered by maintenance (e.g., going from version 3.1.1 to 3.1.2) versus something for which you'll have to pay (e.g., going from version 3 to 4). It's reasonable for you to pay for significant new functionality, assuming that you want it and can use it, but it seems unfair for a vendor to require you to pay for the privilege of going through an upgrade to receive new functionality that you may not want or, in the alternative, risk losing your maintenance.

To minimize their costs of providing support, vendors try to limit the number of releases that they have to support at any one time. They do this by requiring you to upgrade to within a certain release level (e.g., "n–1") or by specifying that support for a previous version will be discontinued some period of time after the next version is released. On the other hand, upgrades are unpleasant and expensive for you. The more frequently you have to upgrade, particularly where an upgrade provides some change in the user experience, the less happy your users will be.

One vendor's standard license states:

> [Vendor] periodically issues software updates that may require additional processing capacity (i.e., CPU memory or additional disk capacity) for Licensee's equipment. Licensee understands this requirement and agrees to purchase equipment as needed to remain current with [Vendor's] software releases. Licensee agrees to install each update and enhancement as soon as reasonably possible but in no event later than ninety (90) days after receipt. In the event that Licensee fails to so install any update or enhancement, then any warranty or obligation of [Vendor] with respect to the affected Program shall be invalidated.

The impact of the quoted language means that if the vendor releases an upgrade that required more processing capacity than your existing hardware had available, then within 90 days of the new release being made available, you *must* (1) upgrade your hardware to the specifications for the upgrade and (2) install the upgrade. Think of the impact that this could have on your budget and your upgrade schedule for your hardware.

Vendors frequently will not commit to a specific number of releases or the frequency with which they will produce new releases. It's not unreasonable to insist on a maximum number of required upgrades in a given year, nor is it unreasonable to expect vendors to keep up with industry-standard hardware/OS upgrades and industry-specific regulations.

It's not unreasonable to insist on a maximum number of required upgrades in a given year.

The Legal Stuff You Usually Ignore

There are whole sections of contracts that business people regularly ignore because they feel like those sections cover stuff that is only interesting to lawyers. Although the lawyers are usually the only ones who argue about these issues, the business people really ought to pay more attention to them. Things like confidentiality, warranties, limitations of liability, and other terms that are always at the back of the contract can have a significant operational and financial impact; frequently, the vendors put things back there assuming that the business people won't be paying close attention.

Confidentiality Provisions

Some vendors' standard licenses define every aspect of their software and documentation to be confidential information. While it is not unreasonable for a vendor's source code or detailed technical documentation to be confidential, making the object code or user manuals confidential could create a problem. For example, one vendor's standard form contract states:

> The parties agree . . . that they shall not use, except as otherwise expressly permitted hereunder, or disclose to any third person, including any of its affiliates, or to any employee of the receiving party without a need to know, either during or after the term of this Agreement, any Confidential Information.

If the definition of "Confidential Information" were to include the object code or user documentation (which this vendor's license does), this would mean that except for employees of your company who actually have a need to see the software, no one else could even see the screen while the software was being used without it being a violation of this term.

Some vendors' standard agreements also have confidentiality provisions that are "one-way": they protect the vendor's information but not yours. If your company is hiring

the vendor to implement the software or provide any other services, the vendor will be learning a great deal about how your company operates and would be under no obligation to keep any of it a secret.

Warranties and Indemnities

High on the list of things that non-lawyers traditionally ignore are the warranties and indemnities. In addition to the warranty that there are no disabling devices included in the software, the big one from your perspective is that the vendor should warrant that the software and documentation do not violate any patents or otherwise infringe on any other intellectual property rights. If someone claims that the software or documentation does violate their intellectual property rights, the vendor should indemnify your company for all costs and expenses associated with that claim.

Limitation of Liability

Next time you drop film off to be developed, look at the disclaimer on the envelope that says that if the developer screws up your pictures or even just completely loses the film, all they have to do is give you a new roll of film. This is a limitation of liability. A limitation of liability provision is a contractual term that specifies the maximum amount of damages for which a party can be liable under the contract.

In most standard vendor licenses, the limitation of liability generally only limits the liability of the vendor and limits it to an amount equal to something like the cost of the software or one year's maintenance. This means, in general, that no matter how badly the vendor screws up or how much it costs your company to fix the problem, the vendor only has to pay your company a maximum of what is specified in this clause.

Whether the cost of the software or the price of a year's maintenance is a reasonable limitation on damages is something that you and your business people can evaluate and negotiate. This provision should also be mutual, so that your company's damages are also capped.

There are certain standard exclusions to this limitation that are generally included in each contract without too much argument from either side. If you're really interested, your general counsel can tell you more.

Other Legal Terms

There are frequently a few other sections at the back of any agreement covering things like "choice of law" and "right of publicity" and other terms that sound trivial, and sometimes they are. However, if you are not careful, your agreement could be governed, for example, by the law of Virginia, which has passed a very vendor-friendly version of the Uniform Computer Information Transactions Act (UCITA).[8] New York is a state that has not passed a version of UCITA and is generally considered to have a well-developed body of commercial law.

Vendors like to publicize their sales. Some companies like to keep very close control over how their name and trademarks are used. Frequently, vendor licenses include the right for the vendor to issue a press release announcing that you are a customer and to use your company's name in their customer list. Your company may have a policy regarding publicity, and if you don't pay attention to the provisions at the end of the license, you could end up agreeing to a contract that violates your company's policy.

Modifications and Customizations

What happens if the software you are buying doesn't do *exactly* what you need it to? The first question is whether you're going to do something about it or just live with the lack of functionality. If you want to do something about it, then you will either need to modify or customize the software. "Modifications" to the software are modules developed by you or a third party that use pre-defined APIs in the software but do not require any changes to the underlying source code. "Customizations" are changes to the underlying source code, made by you, the vendor, or a third party.

Some vendors will encourage you to make modifications to their software, or they may offer to either assist you with developing them or develop them for you. Be careful, though, because some vendors' licenses give them the rights to any modifications you develop.

If you plan to develop customizations, you will need access to the source code for the software. This is not usually provided as part of a standard license. Some companies provide access to source code as part of a "developer's license" that is more expensive than the standard object code license.

Customizations can create risks, though, and the biggest is that a customization will take you out of the upgrade path (this is sometimes true with modifications, too). The best way to make sure that a customization or modification will not lock you into a particular version is to have the vendor develop the customization for you, but negotiating a development and implementation deal is a subject for another time.

Negotiation Tips

The most important thing in any technology procurement is to understand what business need the technology solution must satisfy and what that is worth to the company. Once you understand why you are looking for in a technology solution, the negotiation can begin.

As with any deal, negotiating software licenses is all about who has leverage, and, until you sign the deal, you have the most leverage you are going to have (in some cases, this is still very little, but once you sign the deal you have even less). Frequently, customers are reluctant to negotiate, believing that if they don't make things difficult for the vendor at the negotiating table, the vendor will take that into account during the relationship. It's a nice thought, but this almost never happens. By that logic, you would agree to pay the sticker price for a new car in the hopes that you would get better service from the dealer in the long run. It's important to realize that, in general, a vendor's standard initial position is a position that is very favorable to the vendor. It's simply the opening position in the negotiation. One of the most important aspects of a negotiation is to know what is reasonable and why. Hopefully, this article has given you some insight into that.

Both comparison shopping and getting references are reasonable things to do in any procurement. If there are multiple packages that can provide the functionality that you need, a competitive procurement increases the likelihood that you will get a better deal from whichever vendor you ultimately pick. Also, discussing your needs and a vendor's offering with one vendor will give you ideas for additional questions to ask the other vendor(s). You should also ask the vendor for references, including at least one customer that has stopped using the vendor's software recently. If the vendor has a users group, contact them, as well, and ask them for references similar to your company's

profile and for a former user of the software. References can give you valuable information about issues with the vendor's software, how well the vendor supports the software, and the vendor's plans for the future.

When it comes time to negotiate, most vendors will try to send someone who does not have negotiation authority. He or she will agree to take your positions back to the vendor, particularly to the vendor's legal department, for review. It's the same principle that car dealers use – the salesman actually has no authority and acts like he is on your side, and it's just his manager who's being inflexible and unreasonable. Try to make sure that the person you are dealing with has the authority to agree to changes in the terms of the license. If they don't, then you're wasting your time negotiating with them.

To the extent possible, you should avoid using the vendor's standard form license. Any document can be written any number of ways, and software vendors have had lots of practice at drafting agreements that are interpreted in their favor. If you do have to use the vendor's form, have an experienced attorney look at it to identify areas of concern. Vendors are very good at writing provisions that sound very reasonable but that can have serious consequences to the consumer. You may not be able to do anything about them in negotiations, but at least you will understand the risks of a particular deal.

The common arguments that vendors will make when resisting changes to their form are that the changes will adversely affect their pricing model (i.e., they'll have to charge you more), that they will affect the vendor's ability to recognize the revenue, or that the vendor will have to make special accommodations to manage your contract if you get terms that are different from everyone else. The only one of these that holds any water from the customer perspective is that some provisions of a software license allocate risk (for example, a warranty pushes the risk of a failure onto the vendor), and if you change the allocation of the risk, you may actually legitimately change the assumptions underlying the vendor's pricing. However, vendors tend to use this as an argument much more often than it is merited. If the vendor argues that a change will affect the vendor's ability to recognize the revenue from the sale of the software, ask to see the opinion of the vendor's accountant.

Conclusion

If all of the above wasn't enough to make you want to look through your software licenses with a pair of lawyers and a fine-tooth comb, here's a recent quote from a Microsoft EULA for the patch that fixed a security problem in Windows Media Player:

> You agree that in order to protect the integrity of content and software protected by digital rights management ("Secure Content"), Microsoft may provide security related updates to the OS Components that will be automatically downloaded onto your computer. These security related updates may disable your ability to copy and/or play Secure Content and use other software on your computer. If we provide such a security update, we will use reasonable efforts to post notices on a web site explaining the update.[9]

By downloading and installing that security patch, which you really needed to do, you would give Microsoft the authority to automatically dump software onto your machine, and the only thing they would have to do would be to make a reasonable effort to post a notice about it somewhere on a Web site.[10]

Your software is letting your company do its business, and the licenses are what control how you use that software. Understanding the reasons why your company wants to use a particular software and the terms and conditions governing its use are critical issues if you are going to be involved in negotiating or administering software licenses. At the very least, read the licenses of the software for which you are responsible so that you can have a clear understanding of what you are and aren't allowed to do.

Notes

1. This article provides general information and represents the author's views. It does not constitute legal advice and should not be used or taken as legal advice relating to any specific situation.

2. *http://whatis.techtarget.com/* under EULA *(*as of Sept, 3, 2002).

3. US Code Title 17.

4. PL 105-34 (1998).

5. PL 106-113, 113 Stat 1501, 1501A-521 (1999).

6. Note that there is a difference between "termination for convenience" and "termination for cause." Termination for cause is when the vendor has breached the contract in some way and you are firing the vendor. Termination for convenience means that you have simply decided that you don't want to use the software or the services any more.

7. American Institute of Certified Public Accountants Statement of Position 97-2.

8. VA Code §§ 59.1-501.1 through 59.1-502.1. Maryland is the only other state so far that has passed a version of UCITA, but it is being considered in a number of other states. Three states, Iowa, North Carolina, and West Virginia, have enacted "bomb shelter" anti-UCITA statutes to protect their citizens from the effects of UCITA provisions in shrink-wrap or click-wrap licenses.

9. As quoted in Thomas C. Greene, "MS Security Patch EULA Gives Billg Admin Privileges on Your Box," posted at *http://www.theregister.co.uk/content/4/25956.html*, as of 7/01/02.

10. For a humorous take on this, see J.D. "Illiad" Frazer's comic strip, "User Friendly," for July 6 and 8, 2002, at *http://www.userfriendly.org*.

C99 and compatibility

by Glen McCluskey

Glen McCluskey is a consultant with 20 years of experience and has focused on programming languages since 1988. He specializes in Java and C++ performance, testing, and technical documentation areas.

glenm@glenmccl.com

In previous columns, we've looked at some of the new features in C99, the standards update to C. In this presentation we'll discuss compatibility and look at issues with mixing C89 (the previous C standard) and C99 code. We'll also look at compatibility between C99 and C++.

C99 and C89

Let's start by stating what is probably obvious: if you use new C99 features in your C programming, you should not expect your programs to compile with an older C89 compiler. Here's an example:

```
#include <stdio.h>

struct A {
      int x;
      int y;
};

struct A a = {
      .y = 37,
      .x = 47
};

int main()
{
      printf("%d\n", a.x);
}
```

This program uses the designator feature of C99. When I compile the program as C89, the result is:

```
"e1a.c", line 9: error: expected an expression
      .y = 37,
      ^
"e1a.c", line 10: error: expected an expression
      .x = 47
      ^
2 errors detected in the compilation of "e1a.c".
```

Another example is the use of interspersed declarations and statements:

```
#include <stdio.h>

int main()
{
      int x;
      x = 37;
      int y;
      y = 47;
      printf("%d %d\n", x, y);
}
```

This feature was borrowed from C++ and added to C99.

You can't use C99 features with a C89 compiler, but what about going the other way? What happens if you try to compile a C89 program with a C99 compiler?

For example, consider the following program:

```
#include <stdio.h>

int main()
{
      static x = 37;
      x = g(x);
      printf(""%d\n", x);
}

int g(int x)
{
      return x + 10;
}
```

This usage is legal C89, but not C99. The declaration:

```
static x = 37;
```

leaves off the type (int), and the statement

```
x = g(x);
```

calls an undeclared function. The C99 standard tightened up both of these areas. Requiring that a function be declared before use catches a certain class of errors, such as passing a wrong argument type.

Another example of valid C89 usage that is invalid C99 concerns the use of keywords. For example, restrict is a C99 keyword, so this program is no longer legal:

```
#include <stdio.h>

int restrict = 37;

int main()
{
      printf("%d\n", restrict);
}
```

Other new keywords include inline, _Bool, _Complex, and _Imaginary. There are also many new library functions, which may conflict with existing functions in C89 programs.

A third example is failure to specify a return value:

```
int f()
{
    return;
}

int main()
{
}
```

This is valid C89 but invalid C99.

Beyond a few areas like this, C89 programs should work with a C99 compiler.

C99 and C++

The C++ language was designed on a C base, and in the early days there was emphasis on trying to keep C++ compatible with C, so that C programs could be compiled as C++ code. Since that time, C and C++ have both diverged and converged, and compatibility between them is a complicated issue.

The first point is similar to what we said above about using C99 features with a C89 compiler. There are a great many C++ features that have no equivalent in C99. One example is the C++ template feature:

```
#include <cstdio>

using namespace std;

template <class T> T min(T a, T b)
{
    return a < b ? a : b;
}

int main()
{
    int x = min(37, 47);

    printf("%d\n", x);
}
```

This usage has long been part of C++ but is unknown in C. Another example is function overloading:

```
#include <cstdio>

using namespace std;

void f(int i)
{
    printf("f(int) called\n");
}

void f(double d)
{
    printf("f(double) called\n");
}

int main()
{
    f(37);
}
```

The specific f() to call is determined based on the argument type. Again, there's no C equivalent to this feature.

Just as there are C++ features not known to C, there are C99 features not part of C++. For example, C99 mandates a long long type:

```
#include <stdio.h>

long long x = 0xffffffffffffffffull;

int main()
{
    printf("%llu\n", x);
}
```

Many C++ compilers allow this feature, but if I compile the code using strict conformance compiler options, the result is:

```
"e4a.c", line 3: error: the type "long long" is nonstandard
    long long x = 0xffffffffffffffffull;
                  ^
"e4a.c", line 3: error: the type "long long" is nonstandard
    long long x = 0xffffffffffffffffull;
                  ^
2 errors detected in the compilation of "e4a.c".
```

Another example is the C99 predefined identifier feature, used to obtain the name of a function at compile time:

```
#include <stdio.h>

void f()
{
    printf("%s\n", __func__);
}

int main()
{
    f();
}
```

C and C++ have diverged over the years, but they've also converged in some areas. For example, the following code uses a declaration within a for loop, and is now both legal C (C99) and C++:

```
#include <stdio.h>
int main()
{
```

```
    for (int i = 1; i <= 10; i++)
        printf("%d\n", i);
}
```

Likewise, this code uses //-style comments, an idea C99 borrowed from C++:

```
// This is an example of C++-style comments.

int main()
{
}
```

Another area of incompatibility between C and C++ concerns features that are part of both languages, but which have different semantics. For example, both C and C++ support wide character types, but in C, wchar_t is a typedef defined in a header file, whereas in C++ it is a keyword. Based on this difference, the following code is valid C99, but not C++:

```
int wchar_t = 37;

int printf(const char*, ...);

int main()
{
    printf("%d\n", wchar_t);
}
```

No header file is included in this program, so it's perfectly okay to use wchar_t as an identifier, assuming this is a C99 program. If it's a C++ program, then wchar_t is a keyword, and the program is invalid.

An additional example of different semantics concerns file statics:

```
#include <stdio.h>

static int x = 37;

int main()
{
    printf("%d\n", x);
}
```

This code is legal C and C++, but the C++ usage is deprecated, that is, there is a possibility that the code will not be valid at some future time. The preferred C++ usage is unnamed namespaces:

```
#include <cstdio>

using namespace std;

namespace {
    int x = 37;
}

int main()
{
    printf("%d\n", x);
}
```

Whether this approach is really better than file statics obviously depends on your particular biases.

Conclusions

Suppose that you are concerned about compatibility in a practical way. You might have a large body of C89 code that you are thinking of migrating to C99. Or you might have some C code that you want to compile as C++. What should you do?

It seems likely that current C compilers will be upgraded to incorporate C99 features, and C99 is mostly compatible with existing C89 code. C99 provides some attractive new features that you might want to use. But if you care about compatibility with C++, it's not at all clear if and when C++ will incorporate C99 features. And it seems very unlikely that C will ever adopt many of the distinctive C++ features such as templates.

If you have a body of C code that you compile with a C++ compiler, some of the C99 features will help with compatibility: for example, support for C++-style comments and for mixing declarations and code.

Beyond these basic observations, there is really no alternative to sitting down and identifying the underlying differences between C and C++ and specifying some coding standards for use in your project. For example, if you want to use wide characters in your C application and compile the result with a C++ compiler, then you need to know that C treats wchar_t as a typedef'd type defined in a header, whereas C++ treats it as a keyword.

One Web page that discusses C/C++ differences can be found at *http://david.tribble.com/text/cdiffs.htm*.

practical perl

by Adam Turoff

Adam is a consultant who specializes in using Perl to manage big data. He is a long- time Perl Monger, a technical editor for *The Perl Review*, and a frequent presenter at Perl conferences.

ziggy@panix.com

Lightweight Databases

Many programs need to store some kind of state information between sessions. What's the best way to maintain this data? It depends on the application, of course. This month, I investigate solutions that are easy to use and that fit somewhere in between text files and relational database servers.

Managing Persistent Data

A few months ago, I started writing a program to monitor the online catalogs of several technical publishers periodically. Every time my program would visit a Web site, it would look for recently added book titles. Some of these publishers maintain extensive online catalogs, and as a good Web citizen, I certainly don't want to overload their servers with requests for information my program has already seen. Furthermore, I wanted to highlight new titles to see what I might be interested in reading.

Obviously, my program would need to store some state on disk describing what links had been seen before. However, there are literally dozens of ways to accomplish this, and it's not entirely obvious which one is best.

The lazy programmer's solution would be to write out some flat text files on exit that would be read in the next time my program runs. In this case, flat files would be adequate, if the saved data were relatively simple, such as a list of links stored one per line. However, if I need to save more data (ISBN, title, author, etc.), then other issues arise. For example, I would need to synchronize the functions to read and write to my datafile, to make sure they use the exact same format for both input and output. This might become a little tricky if I need to upgrade my program to store even more information later on.

Another perfectly valid approach is to start out using a relational database engine, such as MySQL, PostgreSQL, or Oracle. This is generally a good choice, except in this situation, using a relational database server seemed like an overengineered solution. I didn't want to get bogged down with details of setting up databases, users, or passwords – I just wanted to write a simple Web crawler and save some data once my program finished. One issue I particularly wanted to avoid was having a program that magically breaks whenever a database server is moved, or when a database user or password changes.

In the end, I found that this simple little program fit into a sweet spot – somewhere between quick-and-dirty flat text files and full relational database servers. Many little programs I've written (most of them just quick hacks) fall into this category. One benefit of using Perl is that I'm not stuck using an inappropriate technology for my problem, whether that technology is overengineered or underengineered.

Of course, in Perl there is more than one way to solve this problem. In this article I want to examine two main types of solutions: the venerable DBM file, and lightweight relational databases.

Persistence Through DBM Files

The classic solution to this data storage problem is the venerable DBM file. DBM files come in many different forms, and all of them can be used to store simple key/value pairs. In this way, DBM files behave much like Perl hash variables, except that the keys and values can be saved and restored for later use.

Using a DBM file is quite simple and requires only a few lines of code to start:

```
#!/usr/bin/perl -w

use AnyDBM_File;
use Fcntl;

my %urls;

tie %urls, "AnyDBM_File", "url_data", O_RDWR | O_CREAT, 0640;

## ... %urls is transparently connected to the file "url_data.db" ...
```

First, we need to load a DBM library. In this case, I loaded the AnyDBM_File library through the use statement on line 3. I then create a new hash variable, %urls, on line 6, and connect that hash to the DBM file url_data.db with the tie statement on line 8. (The default DBM file implementation on my machine adds the .db suffix automatically.) From this point forward, any keys or values that are added or modified to the %urls hash in my program will also be stored on disk in the file url_data.db. The connection will be broken either when my program finishes or when I execute the statement untie %urls;.

All of the magic occurs in the tie statement. This tells Perl to associate the variable %urls with the package AnyDBM_File.

The other parameters are sent to AnyDBM_File to describe the file we wish to use. Here, the parameters are a portion of the filename we'll be using (url_data), the flags used to open the file (O_RDWR | O_CREAT, values that come from the Fcntl module), and the permissions mode (0640, or read/write for the owner, read only for the group).

For my program to monitor online publisher catalogs, I could add a new key to this hash for each URL I process, with the value being the day it was processed. By checking to see if an entry already exists for a particular URL, I can easily identify which URLs are new:

```
foreach (@links) {
    next if defined $urls{$_}; ## We've seen this URL before

    print "New URL: $_\n";
    $urls{$_} = localtime();
}
```

And that's it. The first time I run my little link checker, I'll see a whole slew of URLs fly by. Starting with the second time I run my program, I'll only see the URLs that have been added since the previous run.

Flavors of DBM Files

Using the AnyDBM_File module is guaranteed to work whenever a new DBM file is created, but it does have some problems. Depending on the configuration of your system, Perl will support some of the various implementations of DBM files, including NDBM, Berkeley DB, GDBM, and SDBM. By using AnyDBM, you tell Perl that you don't care which one to use, any one of them is fine. The main problem with AnyDBM is that it is not guaranteed to use the same implementation on two different machines, nor is it guaranteed to open any random DBM file you happen to have. It is quite possible that AnyDBM_File will load the NDBM_File module when you want to open a file created by DB_File or GDBM_File. This operation will fail because the naming conventions or the file formats differ.

Therefore, it is better to choose a specific type of DBM file module instead of the AnyDBM_File:

- NDBM_File uses the native NDBM library on your system, if one exists.
- DB_File uses the Berkeley DB 1.85 library, if present.
- GDBM_File uses the GNU GDBM library, if present.
- SDBM_File is Perl's own DBM library and is always available.

Each of these DBM libraries has its own advantages and disadvantages; see the documentation for AnyDBM_File for more information (man AnyDBM_File or perldoc AnyDBM_File). I use either DB_File or GDBM_File, because they tend to be available on most Perl installations. SDBM_File will always work, and it exists as a DBM implementation of last resort.

Remember that DBM files store simple key/value pairs. If you are programming multi-level data structures, such as a hash of hashes or a hash of lists, then regular DBM files will not store all of your data properly. For these kinds of data structures, look into Joshua Chamas' "multi-level DBM" module, MLDBM, available on the Comprehensive Perl Archive Network (CPAN).

Limits to DBM Files

Perl's support for DBM files makes it easy to add persistent data structures to a program with just a few lines of code. The main disadvantage is that you need to manage all of the data yourself, using Perl hashes. This may be a useful technique in the small, but tends not to scale very well as requirements grow.

Suppose I wanted to create a report program to count URLs, grouped by the day they were first encountered. Using DBM files, that code might look something like this program:

```
#!/usr/bin/perl -w

use strict;

use DB_File;
use Fcntl;

## Load in the cache of URL => date values
my %url_dates;
tie %url_dates, "DB_File", "url_dates", O_RDWR | O_CREAT, 0640;

## Count books (hash entries), grouped by the day they were found
my %count_by_day;
foreach (values %url_dates) {
    ## Strip out the time component of the date
    ## "Sun Aug 11 13:18:59 2002"  -> "Sun Aug 11 2002"
    my $date = $_;
    $date =~ s/\d{2}:\d{2}:\d{2} //;

    $count_by_day{$date}++;
}

## Print out the results (unsorted)
my ($date, $count);
while (($date, $count) = each %count_by_day) {
    print "$date:$count\n";
}
```

This small program re-uses the existing DBM file created by my Web-crawling program that finds new links. Note that this "little" program is 28 lines long (with whitespace and comments). More interesting reports, like one that counts books that contain the word "Perl" in the title, would require more data and might actually be significantly more involved. Now, imagine that two, three, or more of these reports become useful. All of a sudden, the quick-and-dirty solution is starting to run out of

steam, since each new report might require a few dozen lines of new code.

It's clear that DBM files, while useful in some circumstances, aren't always the best or the simplest solution available.

Lightweight Relational Databases

As the requirements for my quick little book-catalog program slowly grow, it's clear that a SQL database is the most appropriate solution, especially if I intend to perform multiple queries on this data. Remember that the problems I intentionally want to avoid are some of the administrative details of setting up databases and passwords with a database engine like MySQL or PostgreSQL. That is, I want my program to "just work," and not be impacted if I happen to move my MySQL server to another computer, convert to PostgreSQL, or change a username or password. Additionally, I want my program to "just work" if I move it to another computer, without requiring that a particular database engine be installed to run this little hack.

Again, we're using Perl, so there's more than one way to do it.

Two ready-to-use modules are available on CPAN that meet my requirements. The first is Jeff Zucker's DBD::CSV module, and the second is Matt Sergeant's DBD::SQLite module. Both of these are database drivers that work with Perl's DBI module, Perl's generic interface to many different database engines. DBD::CSV simulates a relational database by using text files with comma-separated values for each table in the database. DBD::SQLite contains a full-fledged relational database engine written in C that's embedded in the database driver module itself. Neither of these modules require setup, configuration, or a server process to manage the database. They just work.

If you're already familiar with using DBI to connect to MySQL or other relational databases, there is nothing new to learn here. Furthermore, should you need to upgrade from a CSV or a SQLite database, all you need to do is change the DBI connection string, and possibly some of your SQL statements – the rest of your Perl programs remain unchanged.

I used to recommend and use DBD::CSV when I wanted to create a lightweight relational database. Once Matt released his DBD::SQLite module, I started using that instead, since it contains a more robust database engine. This is mostly due to the hard work of Richard Hipp, who created SQLite as a full-featured, embeddable relational database, complete with indexes, transactions, and multiuser access.

Creating a Perl program that uses SQLite is straightforward, assuming you're already familiar with DBI and SQL (another issue entirely):

```
#!/usr/bin/perl -w

use strict;
use DBI;

my $dbname = "url_dates.db";
my $dbh = DBI->connect("dbi:SQLite:dbname=$dbname");

## ... use this SQLite database just like any other DBI database ...
```

One interesting feature of SQLite is that its columns are generally typeless. The column types that are declared in a CREATE TABLE statement are ignored (with the exception of integer primary keys), so there is no need to worry about losing data when storing a 30-character string in a column declared to be of type CHAR(25), or getting an error when storing a string value in an INTEGER column.

Using a relational database makes reporting much easier. For example, a program to count all URLs in the database, grouped by date, would be much simpler than the DBM version seen above:

```
#!/usr/bin/perl -w

use strict;
use DBI;

my $dbh = DBI->connect("dbi:SQLite:dbname=url_dates.db");

my $stmt = $dbh->prepare("SELECT day, COUNT(day)
            FROM urls GROUP BY day");

$stmt->execute();
while (my @row = $stmt->fetchrow_array()) {
    print join(": ", @row), "\n";
}
```

This program is half the size of my previous report program, and all of the logic for this report is contained in the SQL statement on line 4. The while loop at the bottom is reasonably generic and can be abstracted out into a separate sub. It would also be relatively easy to add another SQL query to count the number of books that contain "Perl" in the title – something that would have required more than one extra line of code in the DBM version of the program.

Conclusion

Maintaining persistent data is a common task in Perl programs, and there are easily dozens of ways to do it. For the truly simple tasks, Perl makes simple DBM files available easily and transparently. For more complicated tasks, the easiest solution tends to involve using the DBI, along with a suitable database engine, whether that's something big and powerful, or something small and easy to set up.

generating ethernet packets

by Clif Flynt

Clif Flynt is president of Noumena Corp., which offers training and consulting services for Tcl/Tk and Internet applications. He is the author of *Tcl/Tk for Real Programmers* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

clif@cflynt.com

This article is the first of a series on building network and firewall testing and validation tools using Tcl, open source packages, and some special-purpose hardware. This time I will describe building and testing a Tcl extension for generating Ethernet packets. Subsequent articles will expand on techniques for using this and other extensions.

When I'm building a firewall system I always worry about what I might have missed. Did I install the new security patches in the right places, define the rules correctly, leave no holes?

There are online services like *http://scan.sygatetech.com/* that will scan my system for common flaws, but that requires that I put the system on the Net to test it.

SATAN or SAINT, for example, will check the system for a lot of holes, but they don't test all the firewall rules.

So, I decided to write my own firewall test framework and add new tests as I find and need them.

The first thing I needed for this toolkit was some way to send arbitrary IP packets, to confirm that things like packets on the outside interface with inside addresses are blocked, malformed packets are discarded, and so on.

A little Net searching found a few tools that would almost do what I want. These tools include the Tcl extensions psh from Sun and pkt from USC, and the programs mgen from the Naval Research Laboratory and sendip from Project Purple.

After a little bit of looking, I decided to work with the libdnet library, written by Dug Song (*http://libdnet.sourceforge.net/*). The advantage of this package is that it has the low-level support I need, is currently supported, and has adequate documentation.

The disadvantage is that it's a C library, not a Tcl extension, but that's easily changed.

Tcl was designed to be easily extended. With just a few hours' labor you can pick up a random library and generate the interface code to use it as a Tcl extension. However, this does require some knowledge of how Tcl extensions are constructed. Doing this the first time can take closer to eight hours.

If you don't feel like spending that much time and learning Tcl internals, you can use the SWIG (SoftWare Interface Generator) program to create the interface code for you (*http://www.swig.org/*).

SWIG was developed by David Beazley (*beazley@cs.uchicago.edu*) to make his life easier while he was developing software at Los Alamos. Even in its early forms the program was very useful.

I downloaded the version 1.3.13 for this work. SWIG's built-in support for structures and complex data types is constantly improving. Some details described in this article may be different on the version of SWIG you are using.

SWIG works by examining a definition file that describes the functions and data structures in a library and generating some C code to allow those functions to be loaded into a Perl, Python, Tcl/Tk, Ruby, Guile, or MzScheme interpreter.

Generating a definition file is fairly simple. The basic format is just a list of function declarations.

For example, if you have a file named fibon.c that contains this Fibonacci function:

```
int fib (int i) {
      if (i <= 1) {return 1;}
return fib(i-2) + fib(i-1);
}
```

it could be turned into a Tcl extension with this one-line definition file:

```
$> cat fibon.i
int fib(int i);
```

and this SWIG command line:

```
swig -tcl -module fib -prefix fib -namespace  -v fibon.i
```

The -module fib argument defines the name for this module. The module name can be defined on the command line (as done here) or in the definition file, with the line %module fib.

The -prefix fib argument sets a value that will be used to prevent command name collisions. When used with the -namespace argument, SWIG will generate code to create the new commands in the fib namespace. Placing the extension commands

in a namespace has become the preferred style for Tcl extensions.

Running this SWIG command will create a wrapper file named fibon_wrap.c, which can be compiled into a shared library with a command line resembling this:

```
gcc -shared -L/usr/lib fibon_wrap.c fibon.o -o libfib.so
```

Once this is done, you can load the libfib.so library and use the Fibonacci code in your Tcl scripts just as you would any other Tcl extension.

```
load ./libfib.so
    for {set i 1} {$i &lt; 5} {incr i} {
        puts "The Fibonacci series at level $i is [fib::fib $i]"
    }
```

Unfortunately, most projects are a bit more complex than this.

One problem you run into is that C is a lower level language than Tcl. The C compiler supports data structures that reflect the organization of the data in physical memory, while the Tcl interpreter insulates the programmer from the hardware.

The SWIG solution for this is to generate new Tcl commands for creating and accessing C data structures. The new Tcl commands to create a C data structure will allocate memory for the data structure and return an identifier that Tcl scripts can use to reference the structure. The Tcl script can then pass that identifier to the interface for C functions that need to access the data. As an added benefit, SWIG uses some magic naming conventions to do runtime data checking, so you can't accidentally pass a structure of type a to a function expecting a structure of type b.

You can create an extension with support for creating and using C arrays by adding a little bit of code to the definition file.

The easiest way to do this is to use SWIG's %inline directive. This directive defines functions which should be both included in the final wrapper and exposed to the SWIG parser and code generator.

The SWIG documentation includes this example to show how an array of doubles can be created and accessed:

```
// SWIG helper functions for double arrays
%inline %{
// Create a new array of doubles of a given length
double *new_double(int size) {
    return (double *) malloc(size*sizeof(double));
}
// Delete an array of double
void delete_double(double *a) {
    free (a);
}
// Retrieve the value of an element of the array
double get_double(double *a, int index) {
    return a[index];
}
// Set the value of an element in the array
void set_double(double *a, int index, double val) {
    a[index] = val;
}
%}
```

The new commands can be used like this.

```
# Create an array of doubles
set squares [new_double 5]
# Fill the array with the square of the index
for {set i 0} {$i < 5} {incr i} {
    set_double $squares $i [expr $i * $i]
}
# Invoke a library procedure that requires a
# pointer to an array of doubles as an argument
foo $squares
```

Structures can be a bit more tricky to use but are very simple to describe in the definition file. All you need to do is include the C struct in the body or in an %inline section of the definition file and SWIG will generate a set of interface functions and include them in the Tcl extension.

For example, a structure like this:

```
struct arp {
    unsigned char mac_address[6];
    unsigned char ip_address[4];
}
```

can be accessed with a Tcl script by making a definition file that looks like this:

```
%inline %{
// Define an ARP structure for Machine and IP address
struct arp {
    unsigned char mac_address[6];
    unsigned char ip_address[4];
}
//  A helper utility to set values in an array of
//  unsigned chars -
//  copied from the array example

int unsigned_char_set (unsigned char *ar, \
        int index, unsigned char val) {
    ar[index] = val;
}
//  A test function to display the contents of an
//  arp structure

int showArp (struct arp *p) {
    int i;
    for (i=0; i<6; i++) {
```

```
        printf("0x%x ", p->mac_address[i]);
    }
    printf("\n");
    for (i=0; i<4; i++) {
        printf("0x%x ", p->ip_address[i]);
    }
    printf("\n");
}
%}
```

When SWIG processes this code, it creates these new Tcl commands:

::fib::new_arp *::fib::arp*	Allocates memory for a new arp structure and returns the name to the Tcl script that invokes it.
::fib::delete_arp	Frees the memory associated with an arp structure
::fib::arp_ip_address_get	Returns a handle to access the ip_address C array element of the arp structure.
::fib::arp_mac_address_get	Returns a handle to access the mac_address C array element of the arp structure.
::fib::showArp	An interface into the showArp C function.
::fib::unsigned_char_set	An interface into the unsigned_char_set C function to assign values to elements in a C array.

The Tcl code to test this resembles the following:

```
set arp [arp::new_arp]
set mac [arp::arp_mac_address_get $arp]
for {set i 0} {$i < 6} {incr i} {
    arp::unsigned_char_set $mac $i $i
    }
set ip [arp::arp_ip_address_get $arp]
for {set i 0} {$i < 4} {incr i} {
    arp::unsigned_char_set $ip $i $i
    }

arp::showArp $arp
```

The body of a definition file can usually be extracted from an include file. If you are lucky, you can just use the package's primary include file as a definition file.

The dnet.h file has too much information that's not relevant to creating a wrapper (and is confusing to the SWIG parser), so the simple solution of using dnet.h as a definition file didn't work.

However, all the critical pieces of information (the functions, declarations, and structures used as arguments) are described in the man page, so a set of cut-and-paste operations will create a minimal definition file.

To ensure portability across different word-size machines, the libdnet package uses several data types that aren't part of the basic C language. The SWIG parser doesn't recognize these new datatypes. The SWIG solution for unrecognized data types is to consider them to be pointers.

However, the SWIG parser will recognize a #define or typedef directive to define these datatypes. Adding these lines to the definition file satisfies the SWIG parser:

```
typedef unsigned short uint16_t;
typedef unsigned char  uint_8_t;
typedef unsigned int   uint_32_t;
typedef unsigned int   ip_addr_t;
typedef unsigned int   size_t;
```

To finish the dnet.i definition file, I added versions of the C array access code described above to handle arrays of uint_32_t, uint_16_t, and uint_8_t data.

Once the definition file is complete, SWIG can create a Tcl extension in seconds. The next step is to test the new extension and see if it works.

One of the features of the libdnet library is the ability to send raw packets over the Ethernet. This is as low-level as you can get, and will let me generate whatever type of malformed IP packet I need.

The two critical commands are eth_open, to open a connection to an Ethernet device, and eth_send, to transmit a buffer of binary data (an Ethernet frame).

Syntax: eth_t *eth_open(const char *device);

	Open a connection to an Ethernet device and return a handle for future use.
char *device	The name of the Ethernet device to be connected to, such as eth0, pn0, etc.

Syntax: ssize_t eth_send(eth_t *e, const void *buf, size_t len);

	Transmit a buffer of data over the Ethernet. The buffer should be a valid Ethernet frame. Returns the number of bytes sent. The checksum will be appended automatically.
eth_t *e	The handle returned by eth_open
void *buf	The data to send over the link
size_t len	The number of 8-bit characters to transmit

One problem is that eth_send requires that the buf buffer be a pointer to an area of memory. A Tcl string won't be accepted by the SWIG wrapper. Fortunately, the SWIG wrapper's data validity checking will accept any pointer as a void pointer, so we can use the uint_8_t array commands to create and fill an array of unsigned chars.

Simple code like this will generate garbage packets on the local Ethernet. The data is illegal Ethernet frames, which aren't accepted by other nodes on the network, but running the script will cause the activity lights on an interface card to blink, demonstrating that frames are being sent.

```
# Load the new extension
load ./libdnet.so
# Open a connection to the Ethernet device
set e [dnet::eth_open eth1]
# Create a buffer
set buf [dnet::new_uint_8Array 60]
# Stuff the buffer with incrementing values
for {set i 0} {$i < 60} {incr i} {
    dnet::set_uint_8Array $buf $i $i
}
# And shove it onto the wire 10 times
for {set i 0} {$i < 10} {incr i} {
    dnet::eth_send $e $buf 60

    # Pause for 100 milliseconds
    after 100
}
```

The next step is to send a legal packet and see if it's recognized.

An Ethernet frame consists of five fields of data:

Field size (bytes)	Description
6	The destination MAC address.
6	The source MAC address.
2	A type definition. This is 0x0800 for IP datagrams.
46–1500	The datagram.
4	A Cyclic Redundancy Checksum.

The arp -a command gave me a list of IP addresses and corresponding MAC addresses to fill in the source MAC address and destination MAC address fields; the type field for an IP datagram is 0x0800, and the CRC will be appended by the transmission code.

To generate a valid IP datagram, I used tcpdump with the -x option to get a hex dump of an IP packet. I decided to ping the target node from the node running the Tcl script and grab one of those packets. Using an Echo Request packet provides two sets of validation. Using tcpdump, I can watch the packet arrive on the target node, and I can also see if the target machine responds to the fabricated ping request.

Tcl has full support for operating with lists of data. It makes sense to treat a packet as a Tcl list of hex values until it needs to be converted to an array of unsigned chars for the eth_send command.

The code below creates an Ethernet frame from the various pieces of data. It uses the split command, to convert a colon-delimited MAC address into a list of hex bytes, and the eval command, to combine two lists into a single list.

The Tcl split command will split string data into a list.

Syntax: split *string ?splitChars?*

split	Splits a string into a list. Elements are delimited by a marker character.
string	The string to split.
?splitChars?	A string of characters to mark elements. By default the markers are whitespace characters (tab, newline, space, carriage return). In this example, the character to split on is the colon separating the bytes in a MAC address.

The eval command concatenates the arguments into a string before starting the evaluation. This causes a set of data to lose one level of data grouping. Without eval, a command like lappend list $list2 would be evaluated as lappend list {a b c}, which will append the list element {a b c} to a list. The command eval lappend list $list2 would be evaluated as lappend list a b c, which will append three list elements, a, b, and c to a list.

This script will generate an Ethernet frame and transmit it to the local network:

```
# The MAC address, obtained with arp -s
set destEther 00:E0:4C:00:14:4D
set srcEther 00:A0:CC:D1:B6:00
# A valid echo request packet,
#  obtained with tcpdump
set echo_Request [list 45 00 00 54 00 00 40 00 40 01 \
    05 16 c0 a8 5a 40 c0 a8 5a 02 08 00 98 d9 \
    df 22 00 00 63 cf 4d 3d d5 f3 0e 00 08 09 \
    0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 \
    18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 \
    26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 \
    34 35 36 37]
# Fill a list with hex values
set packet [split $destEther :]
eval lappend packet [split $srcEther :]
```

```
lappend packet 08 00
eval lappend packet $echo_Request
# How many bytes are we using?
set len [llength $packet]
# Create a C array and fill it.
set buf [dnet::new_uint_8Array $len]
for {set i 0} {$i < $len} {incr i} {
   dnet::set_uint_8Array $buf $i 0x[lindex $packet $i]
}
# And shove it onto the wire 10 times
for {set i 0} {$i < 10} {incr i} {
     dnet::eth_send $e $buf $len

     # Wait 100 milliseconds between frames
     after 100
}
```

This extension provides a platform for generating IP packets. The next article will start describing techniques for validating the packet generator before using the generator to validate another system.

USENIX and SAGE Need You

People often ask how they can contribute to our organizations. Here is a list of tasks for which we hope to find volunteers (some contributions not only reap the rewards of fame and the good feeling of having helped the community, but authors also receive a small honorarium). Each issue we hope to have a list of openings and opportunities.

The SAGEwire and SAGEweb staff are seeking:

- Interview candidates
- Short article contributors (see *http://sagewire.sage.org*)
- White paper contributors for topics like these:

Back-ups	Emerging technology	Privacy
Career development	User education/training	Product round-ups
Certification	Ethics	SAGEwire
Consulting	Great new products	Scaling
Culture	Group tools	Scripting
Databases	Networking	Security implementation
Displays	New challenges	Standards
E-mail	Performance analysis	Storage
Education	Politics and the sysadim	Tools, system

- Local user groups: If you have a local user group affiliated with USENIX or SAGE, please mail the particulars to *kolstad@sage.org* so they can be posted on the Web site.

;login: is seeking attendees of non-USENIX conferences who can write lucid conference summaries. Contact Tina Darmohray, *tmd@usenix.org*, for eligibility and remuneration info. Conferences of interest include (but are not limited to): Interop, SOSP, O'Reilly Open Source Conference, Blackhat (multiple venues), SANS, and IEEE networking conferences. Contact *login@usenix.org*.

;login: always needs conference summarizers for USENIX conferences too! Contact Alain Hénon, *ah@usenix.org*, if you'd like to help.

musings,

Or What I Did on My Summer Vacation

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V*.

rik@spirit.com

I once imagined that I would like to spend my life attending conferences. Instead, I am feeling glad to be home, although I am also glad I did get to hang out in a couple of security conferences. And, rather than making you have to drive, ride, or fly, I will share with you parts of my experiences, and something that I think you may find very frightening.

I loathe Las Vegas. Gambling does not appeal to me, so having to walk through three casinos to reach the registration desk at Caesars Palace had me seething inside. I remembered (just in time!) that I am enlightened and cheered up enough to survive the 20-minute check-in line, then another 20-minute wait for the elevator (you'd think this was Eastern Europe, not an expensive hotel), all to attend Black Hat 2002.

The Black Hat conference is designed for security consultants, although I did see DoD types and even some faces from USENIX conferences there. The format consists of three tracks, with intermediate to moderately advanced talks about the security of software and hardware. At the low end, some guys from iDefense gave a lecture about cookies (I liked Kevin Fu's invited talk at last year's Security Symposium better). I enjoyed the explanation of Hogwash and how it had been integrated into Snort codebase as of version 9.2. And how the Honeynet Project plans to proceed with their version 2 honeynets.

Daemon9, now better known as Mark Schiffman of @stake, described his new library, libradiate, which adds to libnet (low-level networking functions for crafting/reading packet headers) with the headers necessary for 802.11B (WiFi). Schiffman demonstrated Omerta, a program that sniffs a wireless channel and disassociates any network card currently associated with an access point. He did not share the source code, a disappointment to many. He did provide other C code examples, but a show of hands revealed that there were only three C coders in the audience. Rather disappointing for a technical con.

While Schiffman rushed through his code examples, FX and Kimo, of Phenoelit (*http://www.phenoelit.de*), were explaining how to turn HP printers into port scanners, using Java code and a class loader included in networked HP printers. The audience found this very amusing (printers scanning a network!), but someone later pointed out to me that HP network printers already will scan networks looking for print servers. What I had missed was their discussion of heap buffer overflows of low-end Cisco routers. Their exploit invalidates the stored configuration and forces a reboot, at which point the router, realizing its configuration is hosed, begins broadcasting a request for a new configuration from anyone. IOS 12 and Cisco 1000, 1600, and 2600 routers are vulnerable to remote attacks, and the 2500 series to local attacks only.

Remember that I mentioned rumors about exploits to IOS in an earlier column. FX made certain that I (and Cisco) understood that they had not done any reverse engineering, just opened the router, recognized that it used a Motorola 68K processor, and used debug messages and information on the Cisco Web site to create their exploit. FX told me that he did not want to be this year's Sklyarov (the Russian arrested at DefCon 9 for explaining how to defeat Adobe's pitiful encryption in eBooks). I really wish that Cisco had succeeded in rewriting IOS as a modern embedded OS instead of abandoning the effort (as far as I have been able to find out).

Hacking has nothing to do with breaking into other people's computers, and everything to do with understanding how things work.

I ran into FX at the next conference I visited, DefCon. DefCon was a bit more subdued this year when compared to previous years, I've been told. I had just enough time to hang out the night before and the first half-day, long enough to get a feel for things and not really missing out on what I hadn't liked about the first DefCon – drunken, chain-smoking teenagers. Keep in mind that DefCon is a serious security conference, with a very low entry fee ($75). Many speakers from Black Hat also speak at DefCon but give a more technical version of their talks.

I listened to Ofir Arkin present the revised version of Xprobe, which does away with the tree structure for probes and instead focuses on a list of probe modules. The modular structure makes Xprobe easier to extend, but it also loses one of the benefits of the original tool, which was accurate TCP/IP fingerprinting with two or three packets. Being an "older guy" who has lost enough hearing (probably from loud concerts in the sixties) was a real disadvantage at DefCon, as two of the conference rooms were outdoor tents, and the roaring of the AC units attempting to keep the temperature at reasonable levels drowned out some of what the speakers said, as well as most questions.

I am sorry I missed Jennifer Granick's talk on the implications of the PATRIOT Act (and yes, it is an acronym) for security practitioners, Simple Nomad's (*http://www.nmrc.org*) talk about the Hacker Nation and how the "War on Terrorism" affects hacking in general, the two lock-picking sessions, and Richard Thieme's (*http://www.thiemeworks.com*) closing session, reminding the audience that hacking is a form of truth seeking.

Lest this last statement confuse you, remember that hacking has nothing to do with breaking into other people's computers and everything to do with understanding how things work – even if it means taking them apart first. My next trip (and why I left DefCon early) took me to San Francisco for the USENIX Security Symposium. The December issue of *;login:* will include the summaries from this conference, as well as other articles dedicated to security, and will be a great edition. I know, as I am the editor and already have some of the articles in hand.

But I don't want to make you wait quite that long. This year's symposium was great, lots of good papers and ITs, and the hall talk was great as well. Professor Felton spoke about the "Freedom to Tinker," another way of saying that reverse engineering of code is akin to US First Amendment rights. Tinkering with things is not only common (would you buy a car where the hood was sealed?) but is good for the community and economically beneficial. Tinkerers have discovered security mistakes in code, and their activities often result in better, competing products (see *http://www.freedom-to-tinker.com/*).

One hall debate led right into a special evening talk about Palladium and TCPA (Trusted Computing Platform Alliance). Tom Perrine, of San Diego Supercomputing Center, asked, rhetorically, why software was so insecure. His argument: that better and more formal design processes would make a huge difference, even when using the insecure programming languages common today (C and Perl as examples). I piped up with my common assertion that you cannot compel people to use formal design processes, but you might instead provide them with safer tools to use – that is, instead of C, using programming languages that enforce good practices, and make it close to impossible to create buffer and heap overflows or to write code that does not check user input, etc. And that this must be implemented on top of a secure operating system that can run untrusted applications in their own compartments. No one agreed

with me, although Perrine did muse about the virtues of ADA, a programming language developed by the DoD for portability and security, and KSOS, an operating system with a trusted kernel.

The EFF's Lucky Green moderated a panel discussion with Peter Biddle of Microsoft and Seth Schoen of the EFF about Palladium. Palladium is Microsoft's project for developing software and hardware for a trusted kernel (see *http://vitanuova.loyalty.org/2002-07-05.html*, under the Microsoft heading, for details). You might think that I would be happy that *someone* is thinking about hardware support in PCs for running a trusted kernel, but the Microsoft focus is of course not the same as an open source focus for security. You should read Ross Anderson's TCPA FAQ for a very detailed critique (*http://www.cl.cam.ac.uk/%7Erja14/tcpa-faq.html*). But I will give you the nutshell here.

Biddle explained how Microsoft's Palladium will work to provide a trusted operating system. Briefly, after booting the trusted kernel, special hardware calculates a hash of this trusted kernel. Then the regular operating system continues with the boot process. The trusted kernel – officially the Trusted Operating Root (TOR), unofficially the "nub" – provides a limited set of services to the operating system and other applications, particularly the ability to seal and unseal "blobs" – any set of data, be it a program, a text file, or a DVD image. The TOR relies on a bit of hardware, a secure cryptographic coprocessor (SCC or SCP) that can perform asymmetric encryption (used in digital signatures) and symmetric encryption (AES in CBC mode), and support a secure store for keys. The SCP also controls memory management, protecting certain regions of memory so that even a root user or the operating system itself cannot access protected memory. At this point, it sounds like there exists the basis for the trusted kernel and compartments that I have long advocated.

But the plan for these wonderful security features is quite different. Instead of protecting the security of your system from attacks, Palladium protects rights of copyright owners. To quote Anderson:

> TCPA and Palladium do not so much provide security for the user as for the PC vendor, the software supplier, and the content industry. They do not add value for the user, but destroy it. They constrain what you can do with your PC in order to enable application and service vendors to extract more money from you. This is the classic definition of an exploitative cartel – an industry agreement that changes the terms of trade so as to diminish consumer surplus.

To provide a few examples of how Palladium and TCPA work to enforce Digital Rights Management (DRM), imagine a system where you cannot migrate files from, say, Microsoft Office 2003 to any other software package. The TOR will not allow you to decrypt the file for the purpose of exporting it to, say, StarOffice. Organizations can configure applications so that data can never be shared, or so that files automatically and irrevocably delete themselves after some time period. No whistle-blowers "leaking" information, no email records detailing dirty deeds, and no more unlicensed copies of Microsoft software, as you *must* have a valid license to run – one that is keyed to your hardware platform using the SCP and the TOR. For the people controlling digital rights, this will be a windfall, as they control how many times you can play a DVD, prevent you from copying it (even by screen scraping), or can even charge you each time you open an application.

Instead of protecting the security of your system from attacks, Palladium protects rights of copyright owners.

We must choose between freedom (with its responsibilities) or passing over control of our computers and aspects of our lives to large corporations.

Microsoft and Intel claim that these new initiatives, once completed, will make your PC more secure, even prevent spam. But, instead of making your own computer more trustworthy for your own use, it will make it trustworthy for the use of content and application providers. Viruses will not be able to affect the TOR or Trusted Agents protected by the TOR, but they will still be able to write to files, delete files, send email – in fact, do almost everything they do today. The only exception will be those files and devices protected by the TOR (which in Palladium includes the keyboard, so no more keystroke sniffers).

Too bad these are DRM initiatives, not real security initiatives. Lucky Green pointed out to me, as does Dar Williams in his July 5 journal entry, that Intel and Microsoft felt they had no choice but to create an unbreakable system for DRM. If they failed to do so, the home entertainment system of the future might not use Intel hardware and Microsoft software. But the very success of these schemes gives each company tremendous leverage, far beyond the virtual monopolies each enjoys today.

You will still be able to run your favorite operating system on Palladium and TCPA-enabled systems. In fact, there were people at the Symposium with IBM T30 laptops that incorporate a TCPA chip. You just won't be able to use any of the features that require the chip, as these make use of a TOR and trusted hardware: for example, an encrypted link to your monitor, DVD-ROM, keyboard, etc.

The RIAA and MPA argue that flagrant copyright violations are destroying their businesses and hurting artists. Perhaps the former might one day be true, but the latter rarely is. Record companies lend money to bands, and it is unusual for the artists that provide the content for RIAA members to make a living as musicians. But providing free digital downloads can help promote artists (read about Janis Ian's experiences as a recording artist and musician, and how free downloads have helped her and Mercedes Lackey, *http://www.janisian.com/article-internet_debacle.html*).

The TCPA chip has been coined the "Fritz" chip, after Senator Fritz Hollings, who has sponsored a law that would make the selling of any computer or storage device that does not support TCPA illegal in the US (*http://www.salon.com/tech/feature/2002/03/29/hollings_bill/*).

The sky is not falling. We find ourselves at a crossroads where we must choose between freedom (with its responsibilities) or passing over control of our computers and aspects of our lives to large corporations. I believe the decision is clear, but I know my mother, as well as many of my friends, doesn't understand the issues (yet). Make yourself heard, ask for real security, and don't give up your freedom.

ISPadmin

Stopping Spam, Part 2

by Robert Haskins

Robert Haskins is currently employed by WorldNET Internet Services, an ISP based in Norwood, MA. After many years of saying he wouldn't work for a telephone company, he is now affiliated with one.

rhaskins@usenix.org

Introduction

This installment of ISPadmin will examine techniques for stopping outbound spam ("unsolicited commercial email," or UCE, originating on your network, destined for a machine or network which you do not control). In the last edition, how to stop spam from the inbound side (from someone else's network to your mailbox) was covered in detail.

Background

Methods for stopping outbound mail are very different from those used to stop inbound spam. Most of the ways outbound spam is stopped can be classified as follows:

- Controlling access to a mail relay machine (e.g., smtp.isp.net)
- Limiting SMTP access to known blocks of open mail relays (e.g., Korea)
- Limiting the number of outbound SMTP connections a client can make over a period of time
- Capping the amount of k/sec an outbound SMTP connection can make

The methods covered in this article will fall into one of the categories listed above, although the coverage will be grouped differently to enable clearer coverage of the topics.

Generic Methods

First, let's discuss generic methods that are not tied directly to a specific open source solution or network hardware (e.g., routers). These methods can be applied to any mail infrastructure, though Sendmail-specific information is listed within this section.

Restricting IP

Controlling what IP addresses are allowed to send mail through a mail server is an important step everyone who runs a mail system on the Internet should take. This is a very common method to control access to a mail relay. In the provider's mail relay machines, a list of IP addresses or blocks is kept that are allowed to relay mail through the relay(s). For Sendmail, the "IP allowed to relay" list is kept in an access database entry similar to the following:

```
209.206.10 RELAY
```

(Sendmail access databases were covered in last issue's ISPadmin column.) Even if you are not a provider, if you are running Sendmail you should be restricting access to your mail relays in this manner. If you don't, you run the very high risk of becoming a spam pariah!

POP before SMTP

The POP before SMTP method requires the end subscriber to simply check their mail before sending it. This method can be used for "roaming" subscribers, who won't be coming from one of the provider's own IP address ranges. Once the POP box is accessed successfully, the subscriber's IP address goes into the IP address "allowed" list on the mail relay(s) for a certain period of time, most commonly 30 minutes. In the case of a Sendmail-based mail relay, the method to control mail relay access can be

Kai's SpamShield is probably one of the oldest packages out there specifically designed to counter outbound spam.

performed via the access database entry, identical to the approach outlined in the "Restricting IP" section.

Mail Message Metering

(Disclaimer: This author developed the Mail Message Metering anti-spam method, which has a patent pending. Describing the method here does not imply the ability to use the system described here.) The Mail Message Metering method is simple in concept and relatively simple to implement. The method is useful to wholesale Internet access providers, although any enterprise that generates lots of outbound mail could use it.

As each subscriber generates an outbound mail message, the network component (switch, RAS gear, DSL aggregating equipment, etc.) redirects the connection to a specially configured mail relay. This specialized mail relay queries a database which contains a current listing of all originating IP addresses that have relayed mail, and associated counts of the number of messages for several time periods (e.g., past minute, past 30 minutes, past hour). If the message would exceed predetermined thresholds, then the message would be re-queued. If the message didn't exceed the limits, then the message would be allowed through and the counts updated appropriately.

Other people and organizations hold anti-spam patents. Of these, Brightmail is probably the best known. However, this author (who is not an attorney) can find no patent (granted or pending) specific to outbound spam.

The benefits of this approach are many:

- Blocks high percentage of outbound spam
- No subscriber and little customer impact
- Configurable and scalable
- Limited impact on authentication (RADIUS) servers

The shortcomings are:

- Requires "white hat" list of legitimate bulk mailers
- Requires use of SMTP redirection (may require additional hardware)

The December 2000 issue of *;login:* contained an in-depth article on the Mail Message Metering solution.

Open Source Packages

One open source package is specifically designed to counter outbound spam (Kai's SpamShield). The others described below can be used to control both inbound and outbound spam.

Kai's SpamShield 1.0

Kai's SpamShield is probably one of the oldest packages out there specifically designed to counter outbound spam. It is a Perl script run out of cron which works by analyzing the most recent sections of the Sendmail log file (usually maillog). The program counts the IP addresses from which messages are originating. If these counts exceed previously entered thresholds, the sender's access to the mail relay is blocked. While dated (it doesn't appear to have been updated since 1997), it is very effective against outbound spam.

Kai's SpamShield version 2.0 was just announced as of this writing in July 2002. No details on the functionality included in the new version exist on the Web site, however.

Blackmail

Blackmail performs various checks against the headers of incoming and outgoing mail messages. These checks include:

- Known sources of spam
- Specific words and/or phrases
- Resolvable names in headers
- Black hole lists
- To: and From: headers
- Correct header formation

While more recent than Kai's SpamShield, it appears that most of these checks are performed by SpamAssassin as well. One difference would be the fact that Blackmail is written in C, while SpamAssassin is written in Perl.

Procmail

System-wide procmail filters can be built to assist in the fight against spam. Two such packages are The SpamBouncer and Email Sanitizer. These work by encapsulating the various anti-spam rule sets (e.g., black hole lookups, resolvable to/from domains, etc.) as procmail recipes. While this author has no direct experience with them, there are enough procmail-based tools out there to indicate this is a valid approach.

SMTP Proxy

SMTP proxies (such as Obtuse Systems Corporation's Juniper firewall toolkit or Trusted Information Systems' fwtk) contain basic SMTP filtering that can be used to control outbound spam. In fact, the Mail Message Metering implementation utilized the Juniper firewall toolkit's smtpd as the basis for the message processing. The proxy approach is a minimalistic one, as SpamAssassin contains much more anti-spam functionality built into it. However, they are implemented in C/C++, which may make the proxies more reliable than code written in Perl.

Stopping Spam at the Network

There are ways spam can be controlled by the provider at the network level:

- Blocking access to known open relays via access control lists (ACLs) on routers
- Caller-ID blocking

The downside to these methods is they do take resources on the network components (such as routers), which can cause additional cash outlays by the provider to implement these methods.

Blocking Access to Known Open Relays

One very effective (but drastic) way to reduce unwanted outbound spam is to simply disallow access to all SMTP servers except for the provider's own mail relays. This could be accomplished by the following ACL on a Cisco router:

```
access-list 101 permit tcp host a.b.c.d any eq smtp access-list 101 permit tcp
    host e.f.g.h any eq smtp
```

System-wide procmail filters can be built to assist in the fight against spam.

Perhaps the most important document a service provider has is its Acceptable Use Policy, or AUP.

```
.
.
.
access-list 101 deny tcp i.j.k.l.0 0.0.0.255 any eq smtp access-list 101 permit
    ip any any
```

The first two access-list statements allow access to legitimate mail relays, and more permit hosts/networks could be added. The third access-list statement denies all other access to port 25 (SMTP) outside of what is specified in the permit list. The final statement allows all other traffic to be routed normally.

A variation on this idea is to block outbound SMTP access to known networks that house open relays, such as Korean networks. A dialup customer should be using the mail relays provided, rather than misconfigured ones located halfway around the world!

Other RAS/Network Techniques

Many spammers will block caller ID to make it harder to track them down. One technique that is used to block spammers from wholesale dialup networks is to disallow outbound SMTP access to anyone who calls in without providing caller ID. This will stop a lot of spam. Also, RAS filters can be loaded dynamically onto the modem ports via RADIUS, allowing SMTP access to a certain set of IP addresses and excluding the rest. In fact, UUNET requires its customers to pass a RADIUS attribute (Ascend-Data-Filter), allowing outbound SMTP access to its wholesale customers' mail relay, and nothing else.

Other tactics that can be attempted include:

- limiting outbound SMTP connection rate
- limiting SMTP bandwidth

This author is not aware where this has been tried "in the wild" on a production network.

Miscellaneous Topics

This section contains odds and ends regarding both inbound and outbound spam.

Acceptable Use Policy

Perhaps the most important document a service provider has is its Acceptable Use Policy, or AUP. Without a properly written AUP, it is impossible to legally shut off customers who abuse a provider's network. All organizations, be they providers, small companies, large companies, nonprofits or others should have an AUP. While it takes time and effort to write a good one, the headache it cures in the long run makes it well worth it.

Legal Aspects

A book could be written on the legal aspects of UCE. In the US the only laws currently governing spam at the federal level surround fax broadcasting (governed by the Federal Communications Commission) and the legality of claims made by spammers (governed by the Federal Trade Commission). Case law is being built every day. In July 2002, Earthlink was awarded US$25 million in a lawsuit against spammers. The FTC has been active in pursuing spammers who make illegal claims.

In the US, the only codified anti-spam law is at the state level. David E. Sorkin has a great site that summarizes the current status of anti-spam law, both inside and outside the US.

Staff

At most ISPs, customer support and/or the network operations center personnel handle spam complaints. At Ziplink, the company dedicated approximately two staff positions to handle the influx of spam complaints, with a 70,000 port dial-in network. Many complaints are duplicates, or are sent in error, which causes additional overhead.

Automated systems such as Spamcop work well. However, they are not infallible and do make mistakes. One benefit of such systems is the elimination of duplication of effort that automated systems can provide. Spamcop will stop sending spam reports to the provider, once the provider tells Spamcop the spammer has been deactivated. However, Spamcop continues to send duplicate spam reports, with the same "footprint" (i.e., source IP address, subject line, etc.) until the provider takes action.

Costs

The additional strain spam puts on staff, machines, and networks is hard to quantify. If we use an assumption that 33% of all email is spam, that loosely translates into 33% higher costs for the provider. Those two additional staff positions mentioned above could be eliminated if spam were not a problem. A server or two could probably be reallocated at a small- to mid-sized ISP, while a larger provider could probably eliminate more. The upstream network connections, if the provider buys transit, would be less without spam.

Usenet News Spam

Most news servers these days are able to control news spammers without much difficulty. InterNetNews (INN) v2.3.2 has an "exponential backoff" feature. The associated control parameters are:

- backoffauth
- backoffdb
- backoffk
- backoffpostfast
- backoffpostslow
- backofftrigger

Check the man pages for inn.conf and search for "backoff" for more information. If the Highwinds Software series (Typhoon/Cyclone/Twister) of news servers is used, a Perl program is available to rate limit article posting. This rate limiting works very well.

Places to Send Your Spam

Ever wonder where you can send spam you receive (besides to the provider that originates it)? A list of email addresses appears below; if anyone knows of additional email addresses to send junk mail to, please send them and they will be published in a future column. Some of these addresses are just statistics trackers, others are for actual complaints, and some are commercial services which block spam using the email submitted to generate rules for protecting their customers.

spamrecycle@chooseyourmail.com	The spam recycling center (statistics)
uce@ftc.gov	FTC's junk mail address
fraud@uspis.gov	For complaints involving US Postal Service mail
enforcement@sec.gov	For securities-related complaints involving US publicly listed companies
cyberfraud@nasaa.org	For securities-related complaints involving North American publicly listed companies
otcfraud@cder.fda.gov	For food/drug-related complaints
junk@brightmail.com	Honeypot address for Brightmail spam filtering service

Conclusion

There are available tools for ISPs (and others) to control outbound spam. Mail transfer agents (MTAs) such as Sendmail can be configured to allow certain IP address ranges to relay mail, which all organizations running a mail server on the Internet today should employ. Outside of MTAs, Kai's SpamShield can be utilized to control outbound spam, and other mail proxy agents can be useful as well. These open source methods work, but are not perfect and take effort to implement. Steps can be taken at the router/network-device level as well, but these are not adaptive and must be regularly updated. Some proprietary methods (such as Mail Message Metering) do exist but are applicable to certain classes of spam sources (such as large ISPs) and are covered by intellectual property law.

References

Blackmail: *http://www.jsm-net.demon.co.uk/blackmail/blackmail.html*

Brightmail: *http://www.brightmail.com/*

Brightmail anti-spam patents: *http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&u=%2Fnetahtml%2Fsearch-adv.htm&r=0&p=1&f=S&l=50&Query=in%2F%22paul%3B+sunil%22%0D%0A&d=ft00*

David E. Sorkin's spam-law site: *http://www.spamlaws.com/*

Email Sanitizer: *http://www.impsec.org/email-tools/procmail-security.html*

Highwinds Software (Typhoon/Cyclone/Twister): *http://www.highwinds-software.com/discussion/index.html*

INN: *http://www.isc.org/products/INN/inn-current.html*

ISP-Planet article on Earthlink spam lawsuit: *http://www.internetnews.com/isp-news/article.php/1430591*

Kai's SpamShield: *http://spamshield.conti.nu/*

Mail Message Metering: *http://www.ziplink.net/ziplink/solutions/mmm/*

Obtuse Systems Juniper firewall toolkit smtpd: *http://www.obtuse.com/smtpd.html*

POP-before-SMTP: *http://popbsmtp.sourceforge.net/*

Relay control in Sendmail for roaming users: *http://www.Sendmail.org/~ca/email/roaming.html*

SpamCon Foundation: *http://www.spamcon.org/*

SpamCon Foundation's list of places to send junk email: *http://www.spamcon.org/recipients/spam-response/help-statistics.shtml*

Spamcop: *http://spamcop.net/*

The SpamBouncer: *http://www.spambouncer.org/*

Trusted Information Systems fwtk: *http://www.fwtk.org/fwtk/*

remote monitoring with SNMP

by Jozef Skvarcek

Jozef Skvarcek is currently working as a system administrator. He holds a PhD in Physics. Computer technology and science are his long-time hobbies.

jozef@photonfield.net

Introduction

It is possible to monitor and administer a small number of computers individually, for example, by running an interactive session on each one. We would be most likely interested in observing the CPU, memory, and disk space utilizations or verifying that a particular process is running. We could run commands like `df` or `ps` at regular intervals in order to assure ourselves that everything is fine. We could also create various smart scripts to perform automatic monitoring functions locally and notify the administrator about the exceptions by, say, email. However, this approach has obvious limitations as the number of servers increases, for it demands a manual modification of the local scripts should the threshold values or the email address change. In general, we would want to have available one centralized point (management station) where we could set up the thresholds and process the notifications about the exceptions. We would prefer to have a unified, simple installation and configuration of the part of the monitoring software which runs on the managed nodes since that would allow for more robust and automated installation procedures for a large number of the systems. Also, we would prefer such a monitoring method to be able to work for the different platforms found in data centers nowadays.

SNMP (Simple Network Management Protocol) fits the task well despite some limitations. It takes care of network communication between the management station and managed nodes. It organizes the management information on the client side so that it can be retrieved and modified by the management station via a small number of SNMP operations. It does not process, filter, or correlate the management information retrieved from the clients. It passes the information to other application programs, or, put differently, the management applications take advantage of SNMP in order to communicate with their clients. One of the compelling reasons for using SNMP is the fact that the SNMP daemon or service is a part of the standard installation of all major modern operating systems. Enterprise-grade database systems, firewalls, and other applications often contain an SNMP module which can communicate the application-specific management information. The components of the network infrastructure such as routers or switches support SNMP; many other devices such as uninterruptible power supplies allow for the installation of a card with an embedded SNMP daemon. Therefore, with relatively little extra burden caused by the planning and configuration, we can have a multi-platform, network-monitoring capability based on open standards.

SNMP standards are defined by the RFC documents created by the Internet Engineering Task Force (IETF). The evolution of SNMP has not been straightforward, varying as different ideas were getting more or less attention. There are three versions of the protocol; the newest (SNMP v3) was standardized on March 27, 2002. From the point of view of the IETF, SNMP is a part of the Internet Standard Management Framework and all three versions share the same basic structure and components:

1. Managed nodes, each with an SNMP entity that provides remote access to management information. These are usually called "agents."
2. One or more SNMP entities with management applications. These are traditionally called "managers."

3. Management protocol to communicate management information between the SNMP entities.
4. Management information.

The architecture of SNMP is modular and it consists of:

1. Data definition language, called Structure of Management Information (SMI). It defines the fundamental data types, object model, and rules for writing the MIB modules.
2. Management Information Base (MIB) modules. They define the objects and event notifications ("traps").
3. Protocol operations. The operations are performed by Protocol Data Units (PDU).
4. Security and administration.

The SMI can be viewed as a collection of management objects residing in a virtual information store, which is the MIB. Collections of related objects are defined in MIB modules written in the SNMP MIB module language. An object in this scheme is called an "Object Identifier" (OID), an ordered sequence of non-negative integers. OID can also be represented by strings; the whole structure resembles DNS. Only a small number of MIBs (the core set) is required to be supported by each conforming agent.

NAME	DEFINITION
MIB-II	RFC 1213
Interfaces MIB	RFC 1573
SNMPv2 Core	RFC 1907

These provide access to a small common set of management information such as the system name, location, contact information and statistics about the network interfaces. There are a large number of non-mandatory MIB modules from the IETF. Another set of MIB modules are written by hardware and software vendors to support the management of their products. These enterprise-specific MIBs specify objects that lie under the "1.3.6.1.4.1" ("iso.org.dod.internet.private.enterprise") branch, sometimes simply called the "enterprise" branch. The administration of the enterprise branch is delegated to the vendors who normally provide their MIB modules as text files on some media, say, CD-ROMs or floppy disks that ship with their products. The administrator can study the files in order to determine which OIDs and traps are available. For example, the administrator would likely be searching for the OIDs holding the current values of input/output voltages or output load if he were reading the MIB which came with a power supply unit.

There are a small number of PDUs; the most frequently used are GET, GET-NEXT, SET, and TRAP. The PDUs are encapsulated into the SNMP messages and then are transmitted over the IP network using the UDP protocol. The SNMP daemon usually listens on port 161 for all PDUs except TRAP, which is sent to port 162. If you have installed a network management application with the GUI such as IBM Tivoli Netview, the PDUs are generated by choosing the appropriate items from the menus. NET-SNMP, the excellent open source SNMP agent, implements the PDUs as command-line executables.

SNMP and Security

The IETF explicitly specifies the following security threats:

1. "Modification of information" – This is the danger of the modification of SNMP information during network transmission by an unauthorized person.
2. "Masquerade" – This is the danger that an unauthorized user may assume the identity of another user with more administrative rights.
3. "Disclosure" – This is the threat that an unauthorized person may observe the SNMP communication between the managed agents and the management station.
4. "Message stream modification" – This is the danger that the SNMP messages can be re-ordered, delayed, or replayed in order to force some unauthorized management operations.

Versions 1 and 2 of the SNMP protocol provided hooks for multiple authentication schemes; however, they did not explicitly specify any but a trivial authentication scheme based on "community" strings. The "community" is included within an SNMP message, the message is transmitted in the clear-text format, and then the community is verified by the receiving entity. Obviously, this security scheme does not really solve the problems posed by the threats and is vulnerable to various sorts of attacks. It is the goal of version 3 to address the threats. It employs the traditional concept of a user who is identified by a username and secret key (passphrase). The security information is associated with the user. The cryptographic algorithms MD5 and SHA are used for the authentication and DES for the encryption. Other protocols are permitted by the standard. The standard also mandates time synchronization between the SNMP entities in order to tackle the message stream modification threat. Note, this time synchronization is performed by the SNMP agent software and is independent of other methods of system-time adjustment such as the NTP. These security methods make SNMPv3 operations safe on the public IP networks. On the other hand, user-based authentication adds another user database not correlated with the system accounts in general. That fact increases the complexity of the administration and implies derived security threats. This problem can be handled by installing an SNMP agent software which supports some kind of centralized and secure user authentication. For example, the latest version of NET-SNMP (version 5) can take advantage of the Kerberos protocol for that purpose.

Which SNMP version to use? SNMPv3 seems to be the trivial answer after what has been said in the previous paragraph. Unfortunately, the standard is too new to be fully supported by all software agents. Your choice will likely by limited by the capabilities of your management station, too. For example, popular enterprise-grade network management applications such as Tivoli Netview or HP OpenView support only SNMPv1 and v2. There are several ways to reduce the risks that stem from running SNMPv1 or v2:

1. Set the community strings. Traditionally, "public" is used as the default for the read-only community and "private" is used for the read-write community. Many agents install with these communities pre-defined. They should be changed before the agent is started for the first time.
2. Disable the write access. In most cases the read-only access is sufficient to satisfy management objectives.
3. Allow PDUs from your management station only. The configuration file of an SNMP agent normally allows you to limit the access only from pre-defined IP

addresses or domain names. If you decide to use domain names, then your DNS server should be safe from unauthorized manipulation of the DNS data.

4. Use a private, closed network for SNMP traffic.

As with other software, security vulnerabilities may be discovered in the code occasionally. Therefore, it is important to keep an eye on the security alerts issued for SNMP software you use and apply the security patches whenever necessary.

We shall take a look at the configuration and utilization of the SNMPv3 protocol in an illustrative case in the following section.

A Practical Example

We used a machine called "Jupiter" as the manager station and a machine called "Europa" as the agent. During preparation of this article, Jupiter was a PC running RedHat Linux 7.2 and Europa was a SPARC box running Solaris 8. NET-SNMP agent software version 4.2.3 was used on both systems. The NET-SNMP software on Jupiter came with the distribution, while the one on Europa was compiled from sources. There were several factors that motivated our use of this particular software. NET-SNMP supports SNMPv3; it is open source, which reduces the cost of the management infrastructure; it is under active development; and it is part of the standard installations of many GNU/Linux distributions. Last but not least, despite the complexity of the SNMP implementation, NET-SNMP has been relatively easy to configure and has been working reliably. It will be assumed in this section that the agent has been properly installed on both systems. (We shall provide more information regarding the installation in the next section.) It is assumed that OpenSSL, required for the encryption, has been installed, too.

Our first objective was to retrieve the system uptime from Europa on Jupiter. How did we know to choose this particular information? Every SNMP agent has to support the core set of MIBs, one of which is MIB-II, defined by RFC 1213. We read the document and found the object sysUpTime with OID 1.3.6.1.2.1.1.3 (iso.org.dod.internet.mgmt.mib-2.system.sysUpTime), described as the time (in hundredths of a second) since the network management portion of the system was last re-initialized. The sysUpTime object is a member of "System Group," which contains other useful objects such as sysName, sysContact, and sysLocation. A MIB may be difficult to read, since the language is meant to be processed by a machine rather than read by a human. (Please consult the books mentioned in the Bibliography section for further information.)

We started by configuring the agent on Europa. We needed to create a user ("john") with a passphrase "secret123" (the passphrase must have at least 8 characters). We put into file /var/ucd-snmp/snmpd.conf the line

```
createUser john MD5 secret123 DES
```

This version of NET-SNMP does not support SHA protocol yet, and therefore we don't really have a choice but to use MD5 for the authentication. It is possible to use a different password for the DES; you would need to put it behind DES on the line. In the next step it is necessary to create the configuration file /usr/local/etc/snmp/snmpd.conf. NET-SNMP uses the View-Based Access Control Model (VACM), defined by RFC 2575. VACM provides the administrator with the capability of allowing a particular user access to only a specified subset of the OID tree at the agent. In our case, we want to give "john" complete read-write access to the management information. Also, we want to enforce SNMPv3 and the highest security level, which means that

"john" will have to use the strong authentication and encryption. Below is an example of the content of a simplified file which satisfies these objectives.

```
# /usr/local/etc/snmp/snmpd.conf
#
# Map the security name into a group name:
#
#       groupName   securityModel   securityName
group  johngrp       usm             john

#
# Create a view for us to let the group have rights to:
#
#       name    incl/excl    subtree     mask(optional)
view   all      included     .1

####
# Finally, grant access to the view.
#
#         group     context  sec.model  sec.level   prefix   read  write    notif
access  johngrp    ""       usm        priv        exact    all     all      none

#
# Set value for `system.sysLocation' object
#
syslocation Datacenter

#
# Set value for `system.sysContact' object
#
syscontact Networking

#
END
#
```

For the sake of brevity, we can't discuss the file in detail; please consult the documentation which comes with the NET-SNMP software. Finally, we are ready to launch the SNMP daemon on Europa with the command:

```
# /usr/local/sbin/snmpd
```

At this point the daemon should be listed among the running processes on Europa. An important transformation to the file /var/ucd-snmp/snmpd.conf occurs during the start-up of the daemon. The createUser line is replaced by john's security key. For the calculation of the key, the daemon used the password and the IP address of Europa. This fact is worth noting, since the security key would need to be re-created should the IP change.

The snmpd is running on Europa, but how can we access the information from Jupiter? We can do it with the GET-NEXT PDU, which is implemented by the command snmpwalk. We can run on Jupiter, for example, the following command:

```
$ snmpwalk -v 3 -u john -l authPriv -a MD5 -A secret123 -x DES \
    -X secret123 europa .iso.org.dod.internet.mgmt.mib-2.system
```

which retrieves the values of the OIDs in the "system" branch. We get several lines as the output and recognize the OIDs defined by MIB-II. Among them are the following:

```
system.sysDescr.0 = SunOS europa 5.8 Generic_108528-12 sun4u
system.sysObjectID.0 = OID: enterprises.ucdavis.ucdSnmpAgent.solaris
system.sysUpTime.0 = Timeticks: (383093) 1:03:50.93
system.sysContact.0 = Networking
system.sysName.0 = europa
system.sysLocation.0 = Datacenter
```

The zero trailing the OID names is the "instance." Some OIDs may have multiple instances; for example, the OID describing the mounting point of a disk partition has as many instances as there are partitions.

One may object that the snmpwalk command is too long and poses a security risk. A local user may learn the passwords by running the ps command while we are executing snmpwalk. To deal with this, we can create the file .snmp/snmp.conf in our home directory with the following content:

```
defVersion 3
defSecurityName john
defAuthType MD5
defAuthPassword secret123
defPrivType DES
defPrivPassword secret123
defSecurityLevel authPriv
```

This file should be readable only for the user since it includes the passwords. Having this file in place, we are able to simplify the snmpwalk command to

```
$ snmpwalk europa .iso.org.dod.internet.mgmt.mib-2.system
```

If we recall our original objective, our task was to retrieve the value of the sysUpTime object. We already have the result; the appropriate line is in the output from snmpwalk. We can get the value of that single object by using the GET PDU, which is implemented by the snmpget command. We can run on Jupiter the following:

```
$ snmpget europa .iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.0
```

which returns as output the line

```
system.sysUpTime.0 = Timeticks: (1015919) 2:49:19.19.
```

Our second objective, which will be less trivial, is to monitor the disk space in the root partition on Europa. The necessary information is provided by Host MIB, defined by RFC 1514. Although this MIB is not mandatory, it is supported by many agents, NET-SNMP among them. The MIB specifies the OIDs under the iso.org.dod.internet.mgmt.mib-2.host branch. The command:

```
$ snmpwalk europa .iso.org.dod.internet.mgmt.mib-2.host
```

produces long output containing much interesting information, such as process names currently running on the system, their arguments, partitions, and mounting points, and so on. We are interested in the latest. Among the output were the following lines:

```
host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageIndex.1 = 1
host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageType.1 = OID:
        host.hrStorage.hrStorageTypes.hrStorageFixedDisk
host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageDescr.1 = /
host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageAllocationUnits.1 =
        4096 Bytes
```

```
host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageSize.1 = 756012
host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageUsed.1 = 120936
```

It makes sense, doesn't it? We should verify that the OIDs really are what they seem by reading their descriptions in the MIB file. There were many instances of these OIDs, but we picked up instance 1 since the OID hrStorageDescr.1 returned "/" and that was what we had been looking for. Then we found the rest of the OIDs with that instance number. The size of the file system could now be calculated by multiplying hrStorage-Size by hrStorageAllocationUnits, and the result would be in bytes. Space utilization as a percentage can be calculated using the formula

```
( hrStorageUsed / hrStorageSize ) * 100.
```

The monitoring can be automated by creating, say, a Perl script which could regularly poll the agent. Such a script could parse a configuration file where we could define the threshold. The script would then fire up an alarm if space utilization greater than the threshold were detected. The procedure could be further scaled up to monitor a large number of systems.

NET-SNMP Compilation Notes

The NET-SNMPv4.2.3 sources were downloaded from the project's home site. The agent compiled without any problem on RedHat Linux 7.2 with GCCv2.96 and on Solaris 8 with Sun Work Shop 6.1. After the sources are unpacked we need to run the "configure" script. The script takes several arguments; you can use:

```
# ./configure —help
```

in order to get the list. SNMPv3 is supported as the default, but the Host MIB is not. It is necessary to include the support by the command-line argument:

```
# ./configure —with-mib-modules="host"
```

The agent can support several more MIBs and other functionality; for example, it is possible to include the support for the tcp_wrappers. The configuration uses /usr/local as the default installation prefix, which is why the snmpd.conf file is located in the /usr/local/etc/snmp directory. It is possible to change the location of this and other NET-SNMP files by choosing the appropriate arguments for the configure script. Please read the included documentation for more information. Then we can compile the binaries:

```
# make
```

and install them with

```
# make install
```

The distribution does not provide provisions for those who wish to create the software packages. There are many Makefile files inside the source tree. One has to manually edit the install targets so that the software will be installed in a directory suitable for the build of the package.

Notes on SNMP Agents on Some Operating Systems

RedHat Linux 7.2

Typically, two packages are installed on a production box from RedHat 7.2 CD media:

- ucd-snmp
- ucd-snmp-utils

A third package ucd-snmp-devel which provides the API, can be installed on a development system. Don't be confused by the names; the packages are the NET-SNMP distribution version 4.2.3. (NET-SNMP had been previously known as UCD-SNMP.) The full path to the configuration file is /etc/snmp/snmpd.conf. The packages from the original CDs contained a security vulnerability and the snmpd daemon did not start under certain circumstances. We replaced them with the latest versions of the RPM packages from the home site of the NET-SNMP project and that fixed both problems.

Solaris

The agent is part of the product called "Solstice Enterprise Agents," which is installed automatically during a typical installation. The product consists of five packages:

- SUNWsacom
- SUNWsasnm
- SUNWsadmi
- SUNWmibii
- SUNWsasdk

where the last one provides the development platform and is not installed if only a runtime environment is required. The configuration files are located in the /etc/snmp/conf directory. However, in our opinion this agent is difficult to configure and the documentation is not sufficient. Also, the Host MIB does not seem to be supported and neither does SNMPv3. Therefore we suggest using NET-SNMP instead.

Windows 2000

The agent is installed automatically during a typical installation. Make sure that the "SNMP Agent Service" is enabled for automatic startup. The agent is configured through the SNMP Agent Service Properties window popup in a manner similar to the other system components. The relevant registry keys are located in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SNMP.

The documentation can be found in the Windows 2000 Server Resource Kit, TCP/IP Core Networking Guide. The agent works well on small systems, but it seems to have certain problems serving the Host MIB information on larger multi-processor boxes. It does not support SNMPv3.

Conclusions

The scope of this article is limited to describing the configuration of the SNMP agent and demonstrating use of management information for the monitoring of the system parameters. The management information served by the Core and the Host MIBs we discussed is sufficient for monitoring:

1. Disk space utilization
2. CPU load
3. Swap space utilization
4. Running processes
5. TCP/IP statistics

and other parameters. The aim is to give the interested reader enough knowledge for starting practical, secure remote monitoring. The other parts of the SNMP are left to the reader for further research.

Although much can be achieved by scripting SNMP polls, as hinted in the paragraph about the monitoring of disk space usage, in many cases it would be desirable to have a GUI-based network management application with a rich set of features. A few commercial products such as IBM Tivoli Netview or HP OpenView, as well as free ones such as GxSNMP, are available, although they do not support SNMPv3 yet.

We did not write much about the TRAP PDU. When a problem comes up, traps are emitted by certain hardware and software components such as network routers, UPSes, and databases that support the feature. The traps are usually processed by a network management application, which can sort them out, perform correlations, and issue notifications. In many cases the traps are more efficient than a periodical polling, since they do not cyclically consume the network bandwidth and host resources.

Acknowledgments

We would like to thank the members of the UniGroup in New York City for valuable suggestions.

Bibliography and Web Sites

BOOKS

Essential SNMP, by D.R. Mauro and K.J. Schmidt, O'Reilly, 2001

Practical Guide to SNMP v3 and Network Management, by D. Zeltserman, Prentice Hall, 1999

SNMP Network Management, by P. Simoneau, McGraw-Hill, 1999

Understanding SNMP MIBs, by D. Perkins and E. McGinnis, Prentice Hall, 1997

WEB SITES

GxSNMP: *http://www.gxsnmp.org/*

HP OpenView: *http://www.openview.hp.com/*

IBM Tivoli Netview: *http://www.tivoli.com/products/index/netview/*

NET-SNMP (UCD-SNMP): *http://net-snmp.sourceforge.net/*

RFC: *http://www.ietf.org/rfc.html*

SNMPv3: *http://www.ibr.cs.tu-bs.de/projects/snmpv3/*

The details of the standardization of SNMPv3: *http://www.ietf.org/IESG/actions.html*

cabling

Just the Tip of the Infrastructure

by James Carlini

James Carlini is president of Carlini & Associates, Inc., a management consulting firm focused on developing marketing strategies and applications of strategic integrated information technology. He offers seminars.

carlini@northwestern.edu
773-370-1888

Cabling problems are common in all buildings and data centers. In performing cabling assessments for more than 15 years for many property management companies and major organizations (as well as acting as an expert witness on several multimillion-dollar lawsuits related to cabling infrastructure problems), I can safely say that your building has cabling problems.

From Wacker Drive to Kansas City to Beverly Hills, I have never walked through a building without finding major problems resulting from a lack of management of the inside wiring and cabling facilities. These problems are going to cost you and your organization a lot of money – maybe even the sustainability of your business.

Huge Insurance Risk Second-Round Funding

Some years ago, most property management firms and large organizations didn't care about cabling problems. They were too busy with more important things, like what type of espresso machine they should be installing in the lunchroom. They would let maintenance go until something drastic happened: overhead cabling troughs would get maxed out; tenants (in properties) would move, leaving abandoned cable under floor cable troughs so that raceways got maxed out; a new application would have to be installed, and dead (abandoned) cable would need to be removed to make room for the new network media.

Today, there are some new rules pertaining to maintaining inside wiring within an organization. Managers in charge have to understand that there is huge liability now if cabling is messed up.

The insurance companies have finally perked up and are pushing the issue of cabling infrastructure. National electrical codes are specifying that abandoned cable must be eliminated and that cable ducts must be free of dead cable.

In liability cases, if a fire spreads due to cabling conduits not being properly fire stopped (covered with a flame-retardant material so the pathway, or riser system, cannot act as a "chimney" for smoke and other toxic fumes to travel through the building), the insurance company can claim that your organization's building was not up to code. No payment. Claim dismissed.

Let's say that the fire caused millions of dollars of damage. Your organization needs the money to restore its capabilities – quickly. The insurance company can wiggle out of paying the claim because they can show that the cabling was not up to the national electrical code as well as the municipality's building code.

Is Your Facility Management Awake?

A fast way to find out if the people responsible for cabling within your facility are "on top of the job" is to ask them if they know about the TWINS in the building.

What are the TWINS? It's a fast view of the health of your cabling, which is the lifeline of your business. TWINS stands for Total pair, Working pair, In-service pair, Non-working pair, and Spare pair.

An example of using TWINS would be to evaluate the amount of spare capacity you have coming to the building. You might think you have a lot of spare capacity because you know you have a 5,000-pair cable coming into the building and you are only using 2,000 of that 5,000.

Here is a quick example of how that assumption can be grossly misleading:

Total Pair: 5,000 – Working Pair: 2,200

In-Service Pair: 2,000 – Non-Working Pair: 3,000 – Spare Pair: 200

(Determine non-working pairs through testing.)

All of a sudden, you realize that you don't have the extra capacity you thought you had. You thought you had more than enough for that new telemarketing department you were going to move into the building that needed 500 pair for their incoming lines. Now it's a crisis. Don't think that the phone company is going to come rescue you.

I have actually seen a building that could not be leased up because they ran out of cable to the building. The phone company diverted "spare cable" to another building being built, and the existing building could not lease up the last 20 percent of the office space. For those who are curious about how long it took for them to run more cable to that building, it was a priority order and it took 18 months. If you or your telecom manager really know the TWINS formula for cabling, you may save yourself a crisis or two. Try asking about this TWINS check with your facilities manager.

Data Centers Are Centers for Potential Disaster

So many organizations have built data centers and call centers on the cheap. The newer centers are sometimes more susceptible to cabling problems because the people in charge didn't spend enough money to do a proper job of insulation, fire stopping, compartmentalizing, and creating a redundant approach.

There are many areas for improvement that are often overlooked because the people in charge may not know what a good data center should be. A great example of the right job is the Chicago E-911 center. Another is AT&T's 10 S. Canal facility, where they have four jet aircraft engines as power back-ups if ComEd fails.

I used to take my students through both because I wanted to show them what "the right way" looked like. These were great field trips for those professionals who were jaded and thought they had seen it all. They were impressed and shocked at the level of reliability and redundancy, but jet aircraft engines can chew up a lot of fuel and are not cheap. Neither is filling up their 330,000-gallon tank (a big price difference compared to a PC surge protector).

Fiber-optic cable and copper cables running through cable trays with the proper loading and strapping gave a real-life example, rather than just talking about a quality standard. No overfilled trays, no spaghetti cable messes or other problems that I saw in so many other buildings – just a perfect design.

Pay Now or Pay Dearly Later

So many organizations have gotten by with substandard design and maintenance on cabling and data centers that they might read this with a "so what" attitude.

Those are the companies that will eventually go out of business or be acquired by their competitors when they hit their first major outage and can't get their data center working, or when they have a fire, a bad design, or disaster and find out they can't collect the insurance because the insurance company sends out some person to review the problem before writing a check for $10 million.

Some of you know-it-alls say that will never happen and insurance companies will always pay out. Sorry to burst your bubble. I know this happens because I was that person, and the check wasn't written.

I have actually seen a building that could not be leased up because they ran out of cable to the building.

the bookworm

BOOKS REVIEWED IN THIS COLUMN

HACKER'S DELIGHT
HENRY S. WARREN, JR.
Boston, MA: Addison-Wesley, 2002. Pp. 306. ISBN 0-201-91465-4.

UNIVERSAL COMMAND GUIDE
GUY LOTGERING & THE UCG TRAINING TEAM
New York: Hungry Minds, 2002. Pp. 1591 + CD-ROM. ISBN 0-7645-4833-6.

ABSOLUTE BSD
MICHAEL LUCAS
San Francisco: No Starch Press, 2002. Pp. 616. ISBN 1-886411-74-3.

SENDMAIL PERFORMANCE TUNING
NICK CHRISTENSON
Boston, MA: Addison-Wesley, 2002. Pp. 228. ISBN 0-321-11570-8.

ECONOMETRICS, VOL. 3.
DALE W. JORGENSON
Cambridge, MA: MIT Press, 2002. Pp. 461. ISBN 0-262-10094-0.

THE COMMUNICATIONS TOOLKIT
P.H. LONGSTAFF
Cambridge, MA: MIT Press, 2002. Pp. 271. ISBN 0-262-12246-4.

THE ESSENTIAL GUIDE TO NETWORKING
JIM KEOGH
Upper Saddle River, NJ: Prentice Hall, 2001. Pp. 407. ISBN 0-13-030548-0.

THE PRACTICAL SQL HANDBOOK, 3RD ED.
J.S. BOWMAN ET AL.
Reading, MA: Addison-Wesley, 1996. Pp. 454. ISBN 0-201-44787-8.

ORACLE WEB APPLICATIONS
ANDREW ODEWAHN
Sebastopol, CA: O'Reilly, 1999. Pp. 256. ISBN 1-565-92687-0.

XML AND SQL SERVER 2000
JOHN GRIFFIN
Indianapolis, IN: New Riders, 2001. Pp. 384. ISBN 0-735-71112-7.

WEB DATABASE APPLICATIONS
HUGH E. WILLIAMS & DAVID LANE
Sebastopol, CA: O'Reilly, 2002. Pp. 582. ISBN 0-596-00041-3.

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He is Chief Knowledge Officer at Matrix NetSystems. He owns neither a dog nor a cat.

peter@matrix.net

This is going to be a strange, disjunctive column.

I was going to rest a bit and write about the SQL books I've been looking at since June. But a number of things have hit my mailbox, and the SQL stuff will just take some space at the end of this piece.

There are five books I really enjoyed this past few weeks. One of them is a little out of line, but I think worthwhile. The others are obvious choices.

Hacking

Hank Warren has been hacking for about 40 years. In that time he has come up with a number of neat ways to do math and some algorithms that are quite useful.

Hacker's Delight is, to me, the computenik's response to W.W. Sawyer's *Mathematician's Delight*, which I discovered when I was in high school. Warren has made the world of elegant and efficient hacking come alive, and has managed to be useful at the same time.

Commands

The Universal Command Guide weighs in at about 4 kilos. It may be the most unreadable book I've got; but it's also the most useful one I've seen.

Every command for Windows 95, 98, Me, NT 4.0, 2000, XP; Solaris 7.8; AIX 4.3.3; OpenBSD 2.7; RedHat Linux 7; NetWare 3.12, 4.11, 5.1.6; Mac OS 9.1; and DOS 6.22 is listed. Over 8000 of them. And they're indexed and cross referenced.

The CD-ROM is very well done, and the indexing is superb. You may not need this tome, but you do need the CD.

FreeBSD

I read most of *Absolute BSD* on my flight home from the security symposium. A very fine piece of work, it isn't about how to implement BSD solutions, but it is about managing systems in situ. Its big lack is that there's no bibliography.

Sendmail

Christenson has not written an ordinary book. This is a book for the sysadmin who spends a lot of time working on her mail servers. UNIX mail servers. All over the world, more mail servers run Sendmail than anything else. About 15 years ago, I looked at the `sendmail.cf` file running on our Sun 3/150. I hope to never have to look at one again.

Featuring Sendmail 8.12, this book may possibly enable me to never look at one again. It is not a replacement for Costales and Allman (1997), but if you meddle with mail servers, you need this volume.

Hard-Core Economics

Dale Jorgenson is a professor of economics at Harvard. But don't let that or the fact that his subject is supposed to be tedious put you off.

Econometrics 3 is the third volume of Jorgenson's papers. It is subtitled "Economic Growth in the Information Age." I was really taken by a number of the articles in this hefty book: "Information Technology and the US Economy" and "Computers and Growth" serve as a great introduction to Jorgenson's interests. "What Ever Happened to Productivity Growth?" and "Did We Lose the War on Poverty?" were really enlightening.

book reviews

Two Other Good Books

While I'm on a roll, I want to tip my hat to P.H. Longstaff, whose interests are in communications business and policy. As we recognize that computers (and the Internet) are a part of that communications business, it becomes ever more necessary to try to understand it. *The Communications Toolkit* does a really good job of providing the flexible strategies we'll need in the future.

Keogh's book is the best introduction to networking for your grandmother I've come across. It is so simple and straightforward that even politicians should be able to comprehend it. At times Keogh dumbs things down a great deal, but this may be necessary.

SQL

After I read the Sleepycat BDB book, I realized that I needed to know more about SQL and its uses. I looked at and read too many items. But here are the four that I found most useful.

Bowman and her colleagues produced their volume over five years ago. Though there are some parts that are dated, I think it may be the very best book I've seen on SQL.

Odewahn's book is also a bit dated, but it is both clear and well written. The early chapters are a bit too introductory, but the final four were very, very good. On occasion, I wanted a bit more detail, though.

Griffin's book is far more up-to-date. It's good, with many useful examples.

Williams and Lane is also good, especially the PHP and MySQL sections. I can recommend all four.

INCIDENT RESPONSE: A Strategic Guide to Handling System and Network Security Breaches

Eugene Schultz and Russell Shumway
New Riders Publishing, ISBN 1-578-70256-9

Reviewed by Anton Chuvakin

Anton Chuvakin, Ph.D., GCIA, is a senior security analyst with a major security company.

anton@netForensics.com

Incident Response by Eugene Schultz and Russell Shumway – the third book with this title that I have reviewed – had to overcome a certain expectation barrier, even though the authors are recognized experts in the security field. It passed the barrier with flying colors, being different but still covering many facets of the intricate incident response (IR) process, with sections on technology, procedures, and, especially, people.

The books starts with security basics. A risk assessment overview with loss estimates and a summary of digital risks (such as privilege escalation, break-in, denial-of-service, etc.) is provided. This material appears to be useful mostly for newcomers to the security field. Formal six-stage incident-response methodology is then presented by the authors: the preparation, detection, containment, eradication, recovery, and follow-up (PDCERF) process helps create a solid skeleton to support the fluid form of the IR process.

Admittedly, the book is less hands-on oriented than some other IR manuals; the reader will not find things like computer forensics-tool command-line options and ext2fs file system internals here. However, the book shines in its coverage of the human aspect of incident response. Written by an ex-CIA Ph.D. psychologist, the amazing chapter on social sciences and incident response covers a diverse range of topics: cybercrime profiling techniques, such as victim counseling and victimology; "modus operandi" identification; attack pattern recognition; establishment of threat level and communication with attackers. The chapter provides an exciting journey into the mind of a computer criminal, a cyber-sleuth, and a cybercrime victim. Also covered are insider attacks, often considered to be the doom of information security. The question "Why do insiders attack?" is thoroughly analyzed. The author overlays social methods on standard incident-response procedure (detection/containment/eradication/recovery) to help understand the crucial role the human element plays in any security incident.

Two chapters are devoted to high-level computer forensics overview. Hard disk basics are explained – FAT, cluster, secure deletion are all given appropriate space. The book goes on to talk about the "guiding principles" of the investigation. A brief overview of forensic software and hardware is also provided but only serves to familiarize the reader with the names of common packages and utilities. For example, TCT coroner kit is only given about 15 lines of text.

Honeypots also take an honorable place in the book. Their role in IR is studied in detail and is deemed important. Honeypots are also tied to PDCERF, and their value in studying attackers, shielding IT resources, and even gathering evidence for court prosecution is recognized. Some common ways of implementing honeypots (such as via virtual environment) are discussed. The authors even digress to touch upon the ethical implication of honeypots.

Another gem is a stimulating chapter on future directions in IR. The ambitious prediction of intelligent automated incident response and attacker tracking tools is made by the authors. While it is known that automated response to security incidents must be viewed with caution, the potential seems to exist for future automated IR "helpers."

book reviews

An overview of legal issues is a must for any IR book. A brief and to-the-point section on US laws and international cybercrime treaties is available.

Last but not least, a short response and reporting checklist is compiled by the authors. It is based on the six-step IR process and will help investigators to structure their efforts and assist with data collection. Also included is a copy of a "Site Security Handbook" (RFC2196), with an extensive list of references.

Overall, the book is an extremely useful guide for security managers and those tasked with organizing/maintaining incident response teams. Skilled computer crime investigators will not learn anything new from this book, but they will likely enjoy the book nevertheless.

HONEYPOTS: TRACKING HACKERS

Lance Spitzner
Addison Wesley Professional,
ISBN: 0-321-10895-7, 480 pp.

Reviewed by Anton Chuvakin

If you liked *Know Your Enemy* by the Honeynet Project, you will undoubtedly like Honeynet Project founder Lance Spitzner's *Tracking Hackers* much more. In fact, even if you did not like *Know Your Enemy*, you will probably be impressed with the new book on honeypots and their use for tracking hackers.

The structure of the book is different from *Know Your Enemy*: Spitzner begins with his first honeypot penetration experience and goes on to talk about all aspects of honeypots. In-depth and structured background on honeypot technology is provided. Honeypots are sorted by the level of interaction with the attacker they are able to provide.

In addition, the book covers the business benefits of using honeypots. By classifying honeypots by their value in the areas of prevention, detection, and response (exactly as done in Honeynet Project white papers), Spitzner analyzes honeypot technology's contribution to an overall security posture. He also describes the differences between research and production honeypots and demonstrates the benefits of both for various deployment scenarios.

A large part of the book is devoted to particular honeypot solutions – "honeyd" by Niels Provos, plus several commercial honeypots – with detailed explanation of how they work. For example, there is a clear description of ARP spoofing and how it is used by the "honeyd" honeypot daemon. An interesting chapter on "homegrown" honeypot solutions (such as the ones used to capture popular worms of 2001) sheds some light on the simplest honeypots that can be built for specific purposes, such as one to capture a popular attack by means of a simple port listener. Use of a UNIX `chroot()` jail environment for honeypots is also analyzed.

Of course, a special chapter is devoted to honeynets, Honeynet Project's primary weapon in the war against malicious hackers. Generation II honeynet technology is introduced in the book. The chapter not only lists honeynet deployment and maintenance suggestions but also talks about the risks of honeynets.

Another great feature of the book is a chapter on honeypot implementation strategies and methods, such as using NAT to forward traffic to a honeypot and DMZ honeypot installation. The information is then further demonstrated using two full honeypot case studies, from planning to operation.

What is even more important, maintaining the honeypot architecture, is covered in a separate chapter. Honeypots are a challenge to run, mainly since no "lock it down and maintain state" is possible. One has to constantly build defenses and hide and dodge attacks that cannot be defended against.

Tracking Hackers also has a "Legal Issues" chapter, written with a lot of feedback from a Department of Justice official. It dispels some of the misconceptions about honeypots, such as the "entrapment" issue, and summarizes wiretap laws and related data-capture problems.

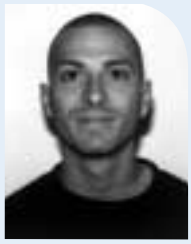
The book is almost the cutting edge of honeypot research and technology; to truly get the cutting edge and learn about the Honeynet Project's latest activities in detail, wait for the second edition of *Know Your Enemy* (coming out next year). In the "Future of Honeypots" chapter, Spitzner includes material on honeypot-based early warning system and distributed deployments, analysis of new threats, expanding research applications, and making honeypots easier to deploy and maintain.

To conclude: Marcus Ranum's enthusiastic preface is not an overstatement. It is indeed a great book, both for security professionals and for others interested in this exciting technology. While I was already familiar with most of the information in the book, it was a fascinating read! This book is a real page turner.

SAGE Elections

by Bryan Andregg

Chair, SAGE 2002
Election Committee



andregg@sage.org

"Autumn approaches
in descending darkness
leading to LISA.

The year's end
means the term's end
and elections
are upon us."

As LISA approaches this year, the end of the term for the current SAGE Executive Committee is upon us. As the Chair for the 2002 Election Committee it is my job to set the schedule for the elections, make sure that a mechanism is in place for them, and make sure that the Nominating Committee provides a suitable slate of candidates. That said, I think my real job is to make sure people get involved, promote candidates, and VOTE! From elections past most members seem to see this last step, the voting, as an artifact of some pre-historic process that they shouldn't bother themselves with.

The schedule for elections is laid out in the SAGE policy document; it was newly amended this year for more flexibility. The key policies summarized are: nominations are open for two months before elections, a candidates forum is scheduled for LISA, elections take place a couple of weeks after LISA, eligibility to vote is set based on the election date. Specific to this year, the schedule is:

Nominations open:	Sept. 1, 2002
Nominations close:	Nov. 22, 2002
Candidates Forum:	LISA, Nov. 7th, 8:00 p.m.
Eligibility Date:	Nov. 22, 2002
Balloting opens:	Dec. 2, 2002
Balloting closes:	Dec. 15, 2002
Results announced:	As soon as practical following balloting

The mechanism for voting will be on-line voting again this year. For those of you who remember, two years ago was our first attempt at on-line voting and it wasn't a resounding success. Operating System and browser compatibility requirements mired an alright system. This year we are implementing a system without those limitations. I will be working with the creator of the on-line balloting system to have it donated so that election costs are kept at a minimum.

The Nominating Committee this year is being led by the estimable Cat Okita. Cat will be assembling a committee in the next couple of weeks (after I write this; it should be together by publication) to solicit and generate nominations. Email with suggested nominations or comments to that committee can be sent to *exec-nom-com@sage.org*.

The SAGE Exec believes in an open election process. So, in addition to the nominating committee the official SAGE policy document also specifies that any five members of SAGE acting together can nominate a candidate. These nominations should be submitted to the SAGE secretary, Trey Harris, by sending email to *trey@sage.org*.

Candidates will be able to address the membership through SAGEwire, the interactive portion of SAGEweb. Members will be able to address questions for candidates through SAGEwire forums and see the responses in the same. Follow-up comments or questions are encouraged.

I am the chair of the election committee, so any questions about what's been in here should be addressed to me at *andregg@sage.org*. That said, don't forget to VOTE!

SAGE membership includes USENIX membership. SAGE members receive all USENIX member benefits plus others exclusive to SAGE.

SAGE members save when registering for USENIX conferences and conferences co-sponsored by SAGE.

SAGE publishes a series of practical booklets. SAGE members receive a free copy of the latest booklet when they join SAGE, and they receive a 33% member discount on all SAGE booklets. In addition SAGE members can freely access the full texts of the booklets on the Web.

SAGE sponsors an in-depth annual survey of sysadmin salaries collated with job responsibilities. Results are available to members online.

The SAGE Web site offers a members-only Jobs-Offered and Positions-Sought Job Center.

SAGE sponsors members-only mailing lists. The archive of the SAGE members list is available on the Web for members only.

SAGE EXECUTIVE DIRECTOR

Rob Kolstad: *kolstad@sage.org*

SAGE MEMBERSHIP

office@sage.org

SAGE ONLINE SERVICES

list server: *majordomo@sage.org*
Web: *http://www.sage.org/*
http://SAGEwire.sage.org
http://SAGEweb.sage.org
http://www.sagecert.org

SAGE STG Executive Committee

PRESIDENT:
David Parter *parter@sage.org*
VICE-PRESIDENT:
Geoff Halprin *geoff@sage.org*
SECRETARY:
Trey Harris *trey@sage.org*
TREASURER
Bryan C. Andregg *andregg@sage.org*
EXECUTIVES:
Tim Gassaway *gassaway@sage.org*
Gabriel Krabbe *gabe@sage.org*
Josh Simon jss@sage.org

USENIX news

Twenty-five Years Ago in ;login:

by Peter H. Salus

USENIX Historian

peter@usenix.org

Yes. It was *;login:* 25 years ago.

In October 1977 the big problems were: being late with *;login:*; back orders on the third distribution tape; lack of contributions for a fourth distribution.

The news was: the new Toronto software release; "You are reminded that a new Toronto release must be submitted even if you had a previous release"; a one-day West Coast meeting in January; and "The membership in the Users' Group now exceeds 250."

In November 1977 there were three major announcements. The first was of the death of Joseph F. Ossanna, the author of NROFF/TROFF. (Joe had given a paper at the May 1977 meeting in Champaign-Urbana.) This was a major loss to the field as a whole and to Bell Labs in particular. (Ten years later, in the October 1987 issue of *;login:*, Jaap Akkerhuis reviewed Emerson & Paulsell, *troff Typesetting for UNIX Systems*. I still use groff/troff.)

The second (also relating to Bell Labs) was that *The C Programming Language* by Ritchie and Kernighan was to be published by Prentice-Hall – at $9.95. Dennis and Brian , if I've not said it often enough, thank you very much.

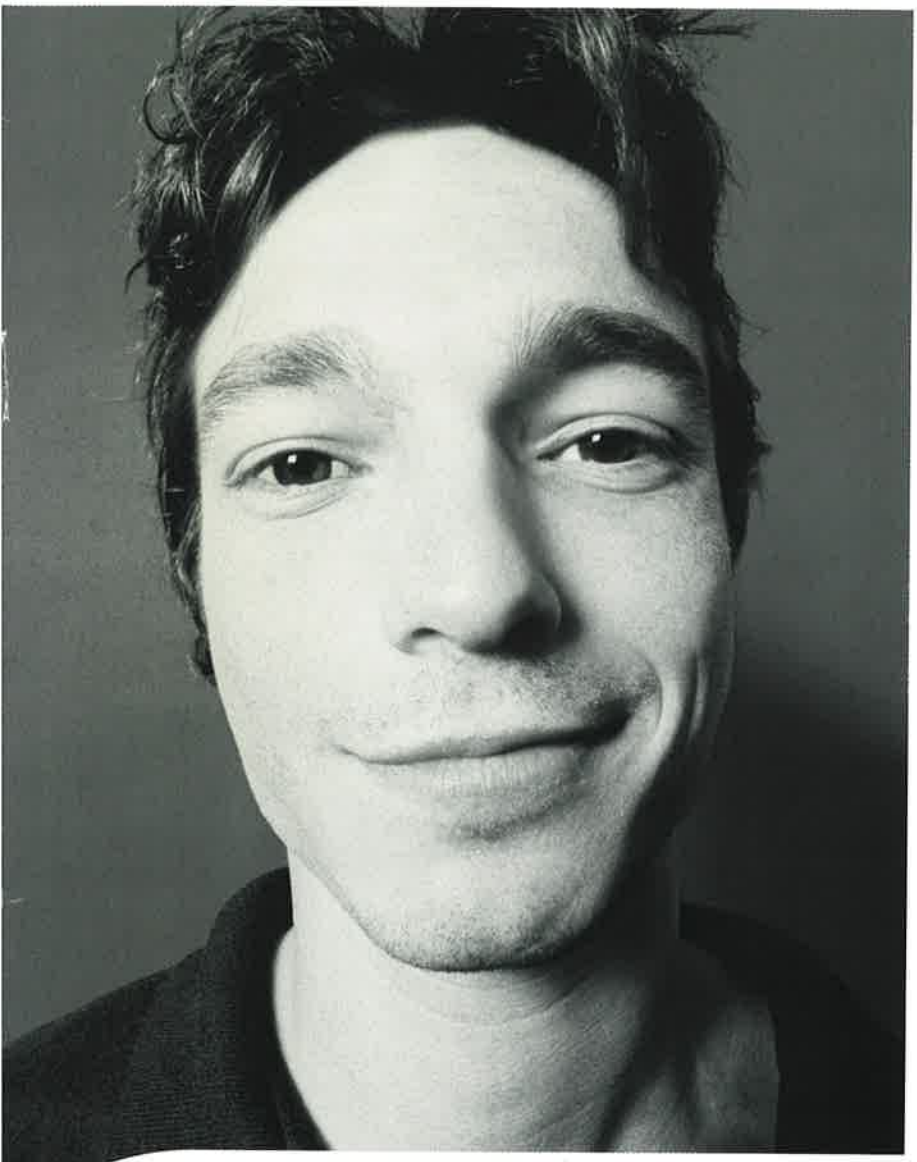
Third, there was to be a meeting (in May) in New York at Columbia University.

It's worth noting that the May 1978 meeting at Columbia, chaired by Lou Katz, was attended by 350 folks, by far the largest meeting of UNIX users up to then.

Oh, yes. The October 1987 issue of *;login:* also carried an article by Charlie Sauer et al. on "RT PC Distributed Services: File System." It really seems long ago!

Finally, in October 1987, USENIX ran two workshops: the Fourth Computer Graphics Workshop in the Cambridge (MA) Marriott (thanks, Tom Duff and, again, Lou Katz); and the POSIX Portability Workshop in the Berkeley (CA) Marina Marriott (thanks, Jim McGinness and, especially, John Quarterman).

I was at both workshops. They were exciting.



"That's *Mr.* SmartyPants to you."

Be the first to know what's happening across your Web systems with SiteScope.®

When it comes to the status of your Web systems, being a "Know-It-All" is a good thing. In fact, it's a critical part of your job. SiteScope from Freshwater Software lets you proactively monitor your network services, system resources, and applications. You're alerted to availability issues and other problems before they can rage out of control. Best of all, **it takes just 20 minutes to install, configure and begin using SiteScope**—no matter how many servers you're monitoring. Looks like people will just have to get used to that look of supreme confidence on your face.

See how easy it can be to proactively monitor your Web systems:

Download a FREE trial of SiteScope today at www.freshwater.com/KnowIt2

Freshwater Software®
a Mercury Interactive company

CONNECT WITH USENIX

MEMBERSHIP AND PUBLICATIONS

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 1+ 510 528 8649
FAX: 1+ 510 548 5738
Email: office@usenix.org
login@usenix.org
conference@usenix.org

WEB SITES

http://www.usenix.org

http://www.sage.org

EMAIL

login@usenix.org

COMMENTS? SUGGESTIONS?

send email to ah@usenix.org

CONTRIBUTIONS SOLICITED

You are encouraged to contribute articles, book reviews, photographs, cartoons, and announcements to *;login:*. Send them via email to *login@usenix.org* or through the postal system to the Association office.

The Association reserves the right to edit submitted material. Any reproduction of this magazine in its entirety or in part requires the permission of the Association and the author(s).

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

;login:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES